WEST

Generate Collection | Print

L4: Entry 15 of 18 File: USPT May 13, 1997

US-PAT-NO: 5630137
DOCUMENT-IDENTIFIER: US 5630137 A

TITLE: Condition handling in a multi-language computer program

DATE-ISSUED: May 13, 1997

INVENTOR-INFORMATION:

NAME	CITY	STATE	ZIP CODE	COUNTRY
Carney; William P.	San Jose	CA		
Conder; Ralph O.	Deerfield Beach	FL		
England; Laurence E.	Morgan Hill	CA		
Grantz; Jeffrey A.	Boca Raton	FL		
Hicks; Daniel R.	Byron	MN		
Lausman; George	North York			CAX
Smith; Robert M.	Morgan Hill	CA		
Tindall; William N. J.	Toronto			CAX

ASSIGNEE-INFORMATION:

NAME	CITY	STATE	ZIP CODE	COUNTRY	TYPE CODE
International Business Machines Corporation	Armonk	NY			02

APPL-NO: 8/ 387494 [PALM]
DATE FILED: February 13, 1995

PARENT-CASE:
RELATED APPLICATION This application is a continuation of U.S. patent application No. 07/755,706, filed Sep. 6, 1991, now abandoned. The present application is related to an application filed on the same date titled "METHOD AND SYSTEM FOR REPRESENTING AND SIGNALING RUN-TIME PROGRAM CONDITIONS" bearing assignee's file number "SA9-91-039", now issued U.S. Pat. No. 5,455,949.

INT-CL: [6] G06 F 9/46

US-CL-ISSUED: 395/678; 395/500
US-CL-CURRENT: 709/108

FIELD-OF-SEARCH: 395/500, 395/650, 395/700

PRIOR-ART-DISCLOSED:

U.S. PATENT DOCUMENTS

Search Selected | Search ALL

	PAT-NO	ISSUE-DATE	PATENTEE-NAME	US-CL
☐	3665421	May 1972	Rehhausser et al.	395/250
☐	3794980	February 1974	Cogar et al.	395/375
☐	4041462	August 1977	Davis et al.	364/775
☐	4128878	December 1978	Yasuhara et al.	395/375
☐	4240137	December 1980	Matsumoto et al.	395/375
☐	4488227	December 1984	Miu et al.	395/375
☐	4564903	January 1986	Guyette et al.	395/411
☐	4733346	March 1988	Tanaka	395/375
☐	4768149	August 1988	Konopik et al.	395/867
☐	4811208	March 1989	Myers et al.	395/800
☐	4835685	May 1989	Kun	395/650
☐	4862351	August 1989	Green et al.	395/375
☐	4980820	December 1990	Youngblood	395/868
☐	4992971	February 1991	Hayashi	395/375
☐	5027273	June 1991	Letwin	395/411
☐	5097533	March 1992	Burger et al.	395/500
☐	5103498	April 1992	Lanier et al.	395/68
☐	5220669	June 1993	Baum et al.	395/775
☐	5305455	April 1994	Anschuetz et al.	395/700

ART-UNIT: 235
PRIMARY-EXAMINER: Lall; Parshotam S.
ASSISTANT-EXAMINER: Vu; Viet
ATTY-AGENT-FIRM: Dawkins; Marilyn Smith

ABSTRACT:

A condition handling method and means capable of handling programs written in a plurality computer programming languages is created by a set of routines which implement the Common Condition Handling (CCH) function. These routines have entry points which are declared as external or entry variables which enables application programs to link to them. The compilers and application programs can then interact with the condition handling process by calling or branching to the entry points in the common condition handling code when the program is executing. Language specific condition handling routines and user condition handlers are invoked using a last-in-first-out (LIFO) queue to associate the handlers with the current stack frame. Any handler may respond to the CCH with one of three types of requests: Resume, Percolate, and Promote. A separate means is provided for registering Exit Handlers and to move the resume cursor.

48 Claims, 9 Drawing figures

L4: Entry 15 of 18 File: USPT May 13, 1997

DOCUMENT-IDENTIFIER: US 5630137 A
TITLE: Condition handling in a multi-language computer program

Detailed Description Paragraph Right (64):
System architectures often limit the availability of some instructions on particular machines. CCH implementations for those machines may choose to provide emulation for these restricted instructions. In those cases, the emulation point is automatic, and execution is resumed with the next sequential instruction (after the point of interrupt) in the program that caused the interrupt. For example, the Condition Manager emulates the extended floating point divide instruction on S/370.

Detailed Description Paragraph Type 1 (104):
Emulate (program checks only)

Detailed Description Paragraph Type 1 (195):
the environmental protection flag (this flag is set and managed by the Condition Manager, it is set upon entry to the Condition Manager (to prevent the overwriting of a ISI associated with a condition that is active within the Condition Manager) and reset upon exit of the condition from active condition handling.)

WEST

Generate Collection | Print

L4: Entry 17 of 18 File: USPT Oct 3, 1995

US-PAT-NO: 5455949
DOCUMENT-IDENTIFIER: US 5455949 A

TITLE: Method for representing and signaling run-time program conditions

DATE-ISSUED: October 3, 1995

INVENTOR-INFORMATION:

NAME	CITY	STATE	ZIP CODE	COUNTRY
Conder; Ralph O.	Deerfield Beach	FL		
Grantz; Jeffrey A.	Boca Raton	FL		
Plaetzer; Scott A.	Rochester	MN		
Smith; Robert M.	Morgan Hill	CA		
Tindall; William N. J.	Toronto			CAX

ASSIGNEE-INFORMATION:

NAME	CITY	STATE	ZIP CODE	COUNTRY	TYPE CODE
International Business Machines Corporation	Armonk	NY			02

APPL-NO: 7/ 755708 [PALM]
DATE FILED: September 6, 1991

INT-CL: [6] G06 F 13/00, G06 F 9/00

US-CL-ISSUED: 395/700; 395/600, 395/375, 395/500, 395/185.02, 364/DIG.1, 364/282.1, 364/286
US-CL-CURRENT: 712/234; 714/49

FIELD-OF-SEARCH: 395/700, 395/500, 395/375, 395/275, 395/600, 371/19, 371/12, 364/DIG.1

PRIOR-ART-DISCLOSED:

U.S. PATENT DOCUMENTS

Search Selected | Search ALL

	PAT-NO	ISSUE-DATE	PATENTEE-NAME	US-CL
☐	4240137	December 1980	Matsumoto	395/375
☐	4488227	December 1984	Miu	395/375
☐	4493034	January 1985	Angelle	395/700
☐	4819233	April 1989	Delucia	371/19
☐	4980820	December 1990	Youngblood	395/275
☐	5103498	April 1992	Lanier	395/68
☐	5175855	December 1992	Putnam	395/700

ART-UNIT: 237
PRIMARY-EXAMINER: Black; Thomas G.
ASSISTANT-EXAMINER: Homere; Jean R.
ATTY-AGENT-FIRM: Dawkins; Marilyn Smith Sterne, Kessler, Goldstein & Fox

ABSTRACT:

An improved method and system is described for generalized handling of conditions occurring during program execution in a computer system having a multi-language Condition Manager (CM). A general signaling routine having object code for an external entry point suitable for linking to application programs written in any language supporting external calls is used. The signaling routine may be used by programs to eliminate the step of checking return codes from subroutines by coding the subroutine to automatically signal the proper condition to the CM which in conjunction with user defined condition handlers takes proper actions in response to the condition. A general condition token which may be used as a feedback token is defined as a condition identifier, a format code for the condition identifier, a severity code for the condition, a control code for a facility identifier, a facility identifier and an optional handle for instance specific information. The signaling routine and the feedback token are used by subroutines which can optionally provide for passing an address, where a feedback token can be stored. The subroutine signals conditions if the severity of the condition is greater than a threshold or else stores a feedback token at the address.

9 Claims, 9 Drawing figures

L4: Entry 17 of 18 File: USPT Oct 3, 1995

DOCUMENT-IDENTIFIER: US 5455949 A
TITLE: Method for representing and signaling run-time program conditions

Detailed Description Paragraph Type 1 (34):
is a 32-bit field containing indicators defined as follows: ##STR1## the application protection flag ##STR2## the environmental protection flag (this flag is set and managed by the Condition Manager, it is set upon entry to the Condition Manager (to prevent the overwriting of a ISI associated with a condition that is active within the Condition Manager) and reset upon exit of the condition from active condition handling.) ##STR3## cross enclave transition flag Note: This flag is set to indicate that the associated condition token and its associated ISI will be passed across an enclave boundary.

Detailed Description Paragraph Type 1 (178):
Emulate (program checks only)

Detailed Description Paragraph Type 1 (179):
System architectures often limit the availability of some instructions on particular machines. CCH implementations for those machines may choose to provide emulation for these restricted instructions. In those cases, the emulation is automatic, and execution is resumed with the next sequential instruction (alter the point of interrupt) in the program that caused the interrupt. For example, the Condition Manager emulates the extended floating point divide instruction on S/370 .

Search Results

Nothing Found

Your search for [flag*<sentence>destruction*<sentence>prevent*] did not return any results.

You may revise it and try your search again below or click advanced search for more options.

flag*<sentence>destruction*<sentenc

SEARCH [**Advanced Search**] [**Search Help/Tips**]

Complete Search Help and Tips

The following characters have specialized meaning:

Special Characters	Description
, ( ) [	These characters end a text token.
= > < !	These characters end a text token because they signify the start of a field operator. (! is special: != ends a token.)
` @ \Q < { [!	These characters signify the start of a delimited token. These are terminated by the end character associated with the start character.

Search Results

Nothing Found

Your search for [flag*<sentence>destroy*<sentence>prevent*] did not return any results.

You may revise it and try your search again below or click advanced search for more options.

flag*<sentence>destruction*<sentenc

SEARCH [**Advanced Search**] [**Search Help/Tips**]

Complete Search Help and Tips

The following characters have specialized meaning:

Special Characters	Description
, ( ) [	These characters end a text token.
= > < !	These characters end a text token because they signify the start of a field operator. (! is special: != ends a token.)
` @ \Q < { [!	These characters signify the start of a delimited token. These are terminated by the end character associated with the start character.

PORTAL

Search DL | flag*<sentence>destroy*<sentence>prevent*

ACM Digital Library

A half century of pioneering concepts and fundamental research have been digitized and indexed in a variety of ways in this special collection of works published by ACM since its inception. The ACM Digital Library includes bibliographic information, abstracts, reviews, and full texts.

Digital Library Overview

- What's New
- DL Pearls
- Content and Organization
- Terms of Usage
- Resources from Affiliated

Browse

- Journ
- Maga
- Trans
- Proc
- News
- Publi
- Spec

Persona

- My Books

Online (

Resources from Affiliated Organizations

Online Services

OCRS

Subscription and Access Information

> Access Information

> Individual Subscriptions

> Institutional Subscriptions

> Document Delivery Service

The ACM Digital Library is published by the Association for Computi

Read the ACM Privacy Policy and Code of Ethics
Questions? Comments? Contact webmaster@acm.org
Call: 1.800.342.6626 (USA & Canada) or +212.626.0500 (Global)
Write: ACM, 1515 Broadway, New York, NY 10036, USA

WEST

Generate Collection | Print

L17: Entry 2 of 46 File: USPT Apr 9, 2002

US-PAT-NO: 6370558
DOCUMENT-IDENTIFIER: US 6370558 B1

TITLE: Long instruction word controlling plural independent processor operations

DATE-ISSUED: April 9, 2002

INVENTOR-INFORMATION:

NAME	CITY	STATE	ZIP CODE	COUNTRY
Guttag; Karl M.	Missouri City	TX		
Read; Christopher J.	Houston	TX		
Balmer; Keith	Bedford			GBX

ASSIGNEE-INFORMATION:

NAME	CITY	STATE	ZIP CODE	COUNTRY	TYPE CODE
Texas Instruments Incorporated	Dallas	TX			02

APPL-NO: 9/ 678746 [PALM]
DATE FILED: October 3, 2000

PARENT-CASE:
This application: is a divisional of U.S. patent application Ser. No. 08/967,102 filed Nov. 11, 1997, now U.S. Pat. No. 6,240,437; which is a divisional of U.S. patent application Ser. No. 08/632,785 filed Apr. 15, 1996 now U.S. Pat. No. 5,742,438; which is a divisional of U.S. patent application Ser. No. 08/160,297 filed Nov. 30, 1993, now U.S. Pat. No. 5,509,129. CROSS REFERENCE TO RELATED APPLICATIONS This application relates to improvements in the inventions disclosed in the following copending U.S. patent applications, all of which are assigned to Texas Instruments: U.S. patent application Ser. No. 08/263,504 filed Jun. 21, 1994, now U.S. Pat. No. 5,471,592 issued Nov. 28, 1995 and entitled MULTI-PROCESSOR WITH CROSSBAR LINK OF PROCESSORS AND MEMORIES AND METHOD OF OPERATION; which is a continuation of U.S. patent application Ser. No. 08/135,754 filed Oct. 12, 1993, now abandoned; which is a continuation of U.S. patent application Ser. No. 07/933,865 filed Aug. 21, 1992, now abandoned; which is a continuation of U.S. patent application Ser. No. 07/435,591 filed Nov. 17, 1989, now abandoned. U.S. patent application Ser. No. 07/437,858 filed Nov. 17, 1989, now U.S. Pat. No. 5,212,777 issued May 18, 1993 and entitled MULTI-PROCESSOR RECONFIGURABLE IN SINGLE INSTRUCTION MULTIPLE DATA (SIMD) AND MULTIPLE INSTRUCTION MULTIPLE DATA (MIMD) MODES AND METHOD OF OPERATION. U.S. patent application Ser. No. 08/264,111 filed Jun. 22, 1994, now U.S. Pat. No. 5,522,083 issued May 28, 1996 and entitled RECONFIGURABLE MULTI-PROCESSOR OPERATING IN SIMD MODE WITH ONE PROCESSOR FETCHING INSTRUCTIONS FOR USE BY REMAINING PROCESSORS; which is a continuation of U.S. patent application Ser. No. 07/895,565 filed Jun. 5, 1992, now abandoned; which is a continuation of U.S. patent application No. 07/437,856 filed Nov. 17, 1989, now abandoned. U.S. patent application Ser. No. 08/264,582 filed Jun. 22, 1994, now U.S. Pat. No. 6,070,003 issued May 30, 2000 and entitled REDUCED AREA OF CROSSBAR AND METHOD OF OPERATION; which is a continuation of U.S. patent application Ser. No. 07/437,852 filed Nov. 17, 1989, now abandoned. U.S. patent application Ser. No. 08/032,530 filed Mar. 15, 1993, now U.S. Pat. No. 6,035,584 issued Mar. 14, 2000 and entitled SYNCHRONIZED MIMD MULTI-PROCESSING SYSTEM AND METHOD; which is a continuation of U.S. patent application Ser. No. 07/437,853 filed Nov. 17, 1989, now abandoned.

U.S. patent application Ser. No. 07/437,946 filed Nov. 17, 1989, now U.S. Pat. No. 5,197,140 issued Mar. 23, 1993 and entitled SLICED ADDRESSING MULTI-PROCESSOR AND METHOD OF OPERATION. U.S. patent application Ser. No. 07/437,857 filed Nov. 17, 1989, now U.S. Pat. No. 5,339,447 issued Aug. 16, 1994 and entitled ONES COUNTING CIRCUIT, UTILIZING A MATRIX OF INTERCONNECTED HALF-ADDERS, FOR COUNTING THE NUMBER OF ONES IN A BINARY STRING OF IMAGE DATA. U.S. patent application Ser. No. 07/437,851 filed Nov. 17, 1989, now U.S. Pat. No. 5,239,654 issued Aug. 24, 1993 and entitled DUAL MODE SIMD/MIMD PROCESSOR PROVIDING REUSE OF MIMD INSTRUCTION MEMORIES AS DATA MEMORIES WHEN OPERATING IN SIMD MODE. U.S. patent application Ser. No. 07/911,562 filed Jun. 29, 1992, now U.S. Pat. No. 5,410,649 issued Apr. 25, 1995 and entitled IMAGING COMPUTER AND METHOD OF OPERATION; which is a continuation of U.S. patent application Ser. No. 07/437,854 filed Nov. 17, 1989, now abandoned. U.S. patent application Ser. No. 07/437,875 filed Nov. 17, 1989, now U.S. Pat. No. 5,226,125 issued Jul. 6, 1993 and entitled SWITCH MATRIX HAVING INTEGRATED CROSSPOINT LOGIC AND METHOD OF OPERATION. This application is also related to the following concurrently filed U.S. patent applications, which include the same disclosure: U.S. patent application Ser. No. 08/160,299, now U.S. Pat. No. 6,116,768 issued Sep. 12, 2000 entitled "THREE INPUT ARITHMETIC LOGIC UNIT WITH BARREL ROTATOR"; U.S. patent application Ser. No. 08/158,742, now U.S. Pat. No. 5,640,578 issued Jun. 17, 1997 entitled "ARITHMETIC LOGIC UNIT HAVING PLURAL INDEPENDENT SECTIONS AND REGISTER STORING RESULTANT INDICATOR BIT FROM EVERY SECTION"; U.S. patent application Ser. No. 08/160,118, now U.S. Pat. No. 6,058,473 issued May 2, 2000 entitled "MEMORY STORE FROM A REGISTER PAIR CONDITIONAL"; U.S. patent application Ser. No. 08/324,323 filed Oct. 17, 1994, now U.S. Pat. No. 5,442,581 issued Aug. 15, 1995 entitled "ITERATIVE DIVISION APPARATUS, SYSTEM AND METHOD FORMING PLURAL QUOTIENT BITS PER ITERATION", which is a continuation of U.S. patent application Ser. No. 08/160,115 filed Nov. 30, 1993 and now abandoned; U.S. patent application Ser. No. 08/159,285, now U.S. Pat. No. 5,596,763 issued Jan. 21, 1997 entitled "THREE INPUT ARITHMETIC LOGIC UNIT FORMING MIXED ARITHMETIC AND BOOLEAN COMBINATIONS"; U.S. patent application Ser. No. 08/160,119, now U.S. Pat. No. 6,016,538 issued Jan. 18, 2000 entitled "METHOD, APPARATUS AND SYSTEM FORMING THE SUM OF DATA IN PLURAL EQUAL SECTIONS OF A SINGLE DATA WORD"; U.S. patent application Ser. No. 08/159,359, now U.S. Pat. No. 5,512,896 issued Apr. 30, 1996 entitled "HUFFMAN ENCODING METHOD, CIRCUITS AND SYSTEM EMPLOYING MOST SIGNIFICANT BIT CHANGE FOR SIZE DETECTION"; U.S. patent application Ser. No. 08/160,296, now U.S. Pat. No. 5,479,166 issued Dec. 26, 1995 entitled "HUFFMAN DECODING METHOD, CIRCUIT AND SYSTEM EMPLOYING CONDITIONAL SUBTRACTION FOR CONVERSION OF NEGATIVE NUMBERS"; U.S. patent application Ser. No. 08/160,112, now U.S. Pat. No. 6,219,688 issued Apr. 17, 2001 entitled "METHOD, APPARATUS AND SYSTEM FOR SUM OF PLURAL ABSOLUTE DIFFERENCES"; U.S. patent application Ser. No. 08/484,113 filed Jun. 7, 1995, now U.S. Pat. No. 5,596,519 issued Jan. 21, 1997 entitled "ITERATIVE DIVISION APPARATUS, SYSTEM AND METHOD EMPLOYING LEFT MOST ONE'S DETECTION AND LEFT MOST ONE'S DETECTION WITH EXCLUSIVE OR", which is a continuation of U.S. patent application Ser. No. 08/160,120; U.S. patent application Ser. No. 08/160,114, now U.S. Pat. No. 5,712,999 issued Jan. 27, 1998 entitled "ADDRESS GENERATOR EMPLOYING SELECTIVE MERGE OF TWO INDEPENDENT ADDRESSES"; U.S. patent application Ser. No. 08/160,116 now U.S. Pat. No. 5,420,809 issued May 30, 1995 entitled "METHOD, APPARATUS AND SYSTEM METHOD FOR CORRELATION"; U.S. patent application Ser. No. 08/160,297, now U.S. Pat. No. 5,509,129 issued Apr. 16, 1996 entitled "LONG INSTRUCTION WORD CONTROLLING PLURAL INDEPENDENT PROCESSOR OPERATIONS"; U.S. patent application Ser. No. 08/159,346, now U.S. Pat. No. 6,067,613 issued May 23, 2000 entitled "ROTATION REGISTER FOR ORTHOGONAL DATA TRANSFORMATION"; U.S. patent application Ser. No. 08/159,652, now abandoned entitled "MEDIAN FILTER METHOD, CIRCUIT AND SYSTEM"; U.S. patent application Ser. No. 08/159,344, now U.S. Pat. No. 5,805,913 issued Sep. 8, 1998 entitled "ARITHMETIC LOGIC UNIT WITH CONDITIONAL REGISTER SOURCE Selection"; U.S. patent application Ser. No. 08/160,301, now U.S. Pat. No. 6,172,305 issued Jan. 9, 2001 entitled "APPARATUS, SYSTEM AND METHOD FOR DIVISION BY ITERATION" U.S. patent application Ser. No. 08/159,650, now U.S. Pat. No. 5,644,522 issued Jul. 1, 1997 entitled "MULTIPLY ROUNDING USING REDUNDANT CODED MULTIPLY RESULT"; U.S. patent application Ser. No. 08/159,349, now U.S. Pat. No. 5,446,651 issued Aug. 29, 1995 entitled "SPLIT MULTIPLY OPERATION"; U.S. patent application Ser. No. 08/482,697 filed Jun. 7, 1995, now U.S. Pat. No. 5,689,695 issued Nov. 18, 1997 entitled "MIXED CONDITION TEST CONDITIONAL AND BRANCH OPERATIONS INCLUDING CONDITIONAL TEST FOR ZERO", a continuation of U.S. patent application Ser. No. 08/158,741 filed Nov. 30, 1993 and now abandoned; U.S. patent application Ser. No. 08/472,828 filed Jun. 7, 1995, now U.S. Pat. No. 5,606,677 issued Feb. 25, 1997 entitled "PACKED WORD PAIR MULTIPLY OPERATION", a continuation of U.S. patent application Ser. No. 08/160,302 filed Nov.

30, 1993 and now abandoned; U.S. patent application Ser. No. 08/160,573, now U.S. Pat. No. 6,098,163 issued Aug. 1, 2000 entitled "THREE INPUT ARITHMETIC LOGIC UNIT WITH SHIFTER U.S. patent application Ser. No. 08/159,282, now U.S. Pat. No. 5,590,350 issued Dec. 31, 1996 entitled "THREE INPUT ARITHMETIC LOGIC UNIT WITH MASK GENERATOR"; U.S. patent application Ser. No. 08/160,111, now U.S. Pat. No. 5,961,635 issued Oct. 5, 1999 entitled "THREE INPUT ARITHMETIC LOGIC UNIT WITH BARREL ROTATOR AND MASK GENERATOR"; U.S. patent application Ser. No. 08/160,298, now U.S. Pat. No. 5,974,539 issued Oct. 26, 1999 entitled "THREE INPUT ARITHMETIC LOGIC UNIT WITH SHIFTER AND MASK GENERATOR"; U.S. patent application Ser. No. 08/159,345, now U.S. Pat. No. 5,485,411 issued Jan. 16, 1996 entitled "THREE INPUT ARITHMETIC LOGIC UNIT FORMING THE SUM OF A FIRST INPUT ADDED WITH A FIRST BOOLEAN COMBINATION OF A SECOND INPUT AND THIRD INPUT PLUS A SECOND BOOLEAN COMBINATION OF THE SECOND AND THIRD INPUTS"; U.S. patent application Ser. No. 08/160,113, now U.S. Pat. No. 5,465,224 issued Nov. 7, 1995 entitled "THREE INPUT ARITHMETIC LOGIC UNIT FORMING THE SUM OF FIRST BOOLEAN COMBINATION OF FIRST, SECOND AND THIRD INPUTS PLUS A SECOND BOOLEAN COMBINATION OF FIRST, SECOND AND THIRD INPUTS"; U.S. patent application Ser. No. 08/426,992 filed Apr. 24, 1995, now U.S. Pat. No. 5,493,542 issued Feb. 20, 1996 entitled "THREE INPUT ARITHMETIC LOGIC UNIT EMPLOYING CARRY PROPAGATE LOGIC", which is a continuation of U.S. patent application Ser. No. 08/159,640 now abandoned; and U.S. patent application Ser. No. 08/160,300, now U.S. Pat. No. 6,026,484 issued Feb. 15, 2000 entitled "DATA PROCESSING APPARATUS, SYSTEM AND METHOD FOR IF, THEN, ELSE OPERATION USING WRITE PRIORITY."

INT-CL: [7] G06 F 15/00

US-CL-ISSUED: 708/603
US-CL-CURRENT: 708/603

FIELD-OF-SEARCH: 708/524, 708/490, 708/625, 708/603

PRIOR-ART-DISCLOSED:

U.S. PATENT DOCUMENTS

Search Selected | Search ALL

	PAT-NO	ISSUE-DATE	PATENTEE-NAME	US-CL
☐	4985848	January 1991	Pfeiffer et al.	
☐	6078941	June 2000	Jiang et al.	708/625

ART-UNIT: 2121
PRIMARY-EXAMINER: Mai; Tan V.
ATTY-AGENT-FIRM: Marshall, Jr.; Robert D. Brady, III; W. James Telecky, Jr.; Frederick J.

ABSTRACT:

A data processing apparatus including a multiplier unit forming a product from L bits of each two data buses of N bits each N is greater than L. The multiplier forms a N bit output having a first portion which is the L most significant bits of the of product and a second portion which is M other bits not including the L least significant bits of the product, where N is the sum of M and L. In the preferred embodiment the M other bits are derived from other bits of the two input data busses, such as the M other bits of the first input data bus. An arithmetic logic unit performs parallel operations (addition, subtraction, Boolean functions) controlled by the same instructions. This arithmetic logic unit is divisible into a selected number of sections for performing identical operations on independent sections of its inputs. The multiplier unit may form dual products from separate parts of the input data. A

single instruction controlling both the multiplier unit and the arithmetic logic unit permits addition of dual products. The dual products are temporarily stored in a data register permitting the multiply and add operations to be pipelined. The dual products are formed in one data word and added by a rotate/mask and add operation in a three input arithmetic unit.

32 Claims, 70 Drawing figures

L17: Entry 2 of 46 File: USPT Apr 9, 2002

DOCUMENT-IDENTIFIER: US 6370558 B1
TITLE: Long instruction word controlling plural independent processor operations

Detailed Description Paragraph Right (398):
The "N C V Z" field (bits 28-25) indicates which bits of the status are protected from alteration during execution of the instruction. The conditions of the status register are: N negative; C carry; V overflow; and Z zero. If one or more of these bits are set to "1", the corresponding condition bit or bits in the status register are protected from modification during execution of the instruction. Otherwise the status bits of status register 210 are set normally according to the resultant of arithmetic logic unit 230.

WEST

Generate Collection | Print

L17: Entry 5 of 46 File: USPT Jan 9, 2001

US-PAT-NO: 6173394
DOCUMENT-IDENTIFIER: US 6173394 B1

TITLE: Instruction having bit field designating status bits protected from modification corresponding to arithmetic logic unit result

DATE-ISSUED: January 9, 2001

INVENTOR-INFORMATION:

NAME	CITY	STATE	ZIP CODE	COUNTRY
Guttag; Karl M.	Missouri City	TX		
Poland; Sydney W.	Katy	TX		
Balmer; Keith	Bedford			GBX

ASSIGNEE-INFORMATION:

NAME	CITY	STATE	ZIP CODE	COUNTRY	TYPE CODE
Texas Instruments Incorporated	Dallas	TX			02

APPL-NO: 9/ 372470 [PALM]
DATE FILED: August 11, 1999

PARENT-CASE:
This application is a division of Application No. 08/160,118 filed Nov. 30, 1993, now U.S. Pat. No. 6,058,473.

INT-CL: [7] G06 F 9/308, G06 F 9/318

US-CL-ISSUED: 712/226; 712/221, 712/224, 708/525
US-CL-CURRENT: 712/226; 708/525, 712/221, 712/224

FIELD-OF-SEARCH: 712/220, 712/221, 712/223, 712/224, 712/225, 712/226, 708/525

PRIOR-ART-DISCLOSED:

U.S. PATENT DOCUMENTS

Search Selected | Search ALL

	PAT-NO	ISSUE-DATE	PATENTEE-NAME	US-CL
☐	4785393	November 1988	Chu et al.	712/221

ART-UNIT: 278
PRIMARY-EXAMINER: Vu; Viet D.
ATTY-AGENT-FIRM: Marshall, Jr.; Robert D. Brady, III; W. James Telecky, Jr.; Frederick

J.

ABSTRACT:

A data processing apparatus includes plural data registers, an arithmetic logic unit and a status register. The status register stores a plurality of different types of status bits. These status bits could be a negative status bit, a carry status bit, an overflow status bit and a zero status bit. These status bits are normally set dependent upon the condition of the result generated by the current arithmetic logic unit operation. A status bit protect instruction type permits selection of status bits protected from modification corresponding to the current arithmetic logic unit result. This status bit protect instruction preferably includes individual protect bit corresponding to each status bit. If a protect bit has a first digital state, then the corresponding status bit may be modified corresponding to the current arithmetic logic unit result. If the protect bit has a second opposite digital state, then the corresponding status bit is protected from modification according to the arithmetic logic unit results.

19 Claims, 71 Drawing figures

WEST

Generate Collection | Print

L17: Entry 24 of 46 File: USPT Dec 9, 1997

US-PAT-NO: 5696959
DOCUMENT-IDENTIFIER: US 5696959 A

TITLE: Memory store from a selected one of a register pair conditional upon the state of a selected status bit

DATE-ISSUED: December 9, 1997

INVENTOR-INFORMATION:

NAME	CITY	STATE	ZIP CODE	COUNTRY
Guttag; Karl M.	Missouri City	TX		
Poland; Sydney W.	Katy	TX		
Balmer; Keith	Bedford			GB2

ASSIGNEE-INFORMATION:

NAME	CITY	STATE	ZIP CODE	COUNTRY	TYPE CODE
Texas Instruments Incorporated	Dallas	TX			02

APPL-NO: 8/ 478129 [PALM]
DATE FILED: June 7, 1995

PARENT-CASE:
This is a continuation of application Ser. No. 08/160,118, filed Nov. 30, 1993.

INT-CL: [6] G06 F 9/312

US-CL-ISSUED: 395/595; 395/800, 395/566
US-CL-CURRENT: 712/245; 712/225

FIELD-OF-SEARCH: 395/375, 395/800, 395/416, 395/595, 395/566, 364/763

PRIOR-ART-DISCLOSED:

U.S. PATENT DOCUMENTS

Search Selected | Search ALL

	PAT-NO	ISSUE-DATE	PATENTEE-NAME	US-CL
☐	4449184	May 1984	Pohlman, III	395/250
☐	4692891	September 1987	Yamaoka et al.	364/763
☐	4747046	May 1988	Baum et al.	395/375
☐	4873627	October 1989	Baum et al.	395/375
☐	5313648	May 1994	Ehlig et al.	395/800

ART-UNIT: 235
PRIMARY-EXAMINER: Ellis; Richard L.
ASSISTANT-EXAMINER: Patel; Gautam R.
ATTY-AGENT-FIRM: Marshall, Jr.; Robert D. Kesterson; James C. Donaldson; Richard L.

ABSTRACT:

A memory store operation comes from one of a pair of registers selected by an arithmetic logic unit condition. An instruction logic circuit (250, 660) controls an addressing circuit (120) to store data in a first register into memory if a selected status bit has a first state and to store data in a second register associated with the first register into memory if the selected status bit has a second state in response to a register pair conditional store instruction. The bits may indicate a negative output of the arithmetic logic unit (230), a carry out signal, an overflow, or a zero output. The register pair conditional store instruction designates a particular one of the status bits to control the conditional store. The instruction logic circuit (250, 660) substitutes the selected status bit for a least significant bit of the register number. Thus memory store is from the first register if the status bit is "1" and is from the second register if the status bit is "0". In a further embodiment the register pair conditional write instruction is conditional. The write operation aborts if the designated condition is true. In the preferred embodiment of this invention, the arithmetic logic unit (230), the status register (210), the data registers (200) and the instruction decode logic (250, 660) are embodied in at least one digital image/graphics processor (71) as a part of a multiprocessor formed in a single integrated circuit (100) used in image processing.

52 Claims, 72 Drawing figures

L17: Entry 24 of 46 File: USPT Dec 9, 1997

DOCUMENT-IDENTIFIER: US 5696959 A
TITLE: Memory store from a selected one of a register pair conditional upon the state of a selected status bit

Detailed Description Paragraph Right (394):
The "N C V Z" field (bits 28-25) indicates which bits of the status are protected from alteration during execution of the instruction. The conditions of the status register are: N negative; C carry; V overflow; and Z zero. If one or more of these bits are set to "1", the corresponding condition bit or bits in the status register are protected from modification during execution of the instruction. Otherwise the status bits of status register 210 are set normally according to the resultant of arithmetic logic unit 230.

Detailed Description Paragraph Right (404):
The "N C V Z" field (bits 28-25) indicates which bits of the status are protected from alteration during execution of the instruction. The conditions of the status register are: N negative; C carry; V overflow; and Z zero. Status register protection circuit 261 receives these bits from instruction word 255 and the negative, carry, overflow and zero status indicators from arithmetic logic unit 230. If one or more of the bits of "N C V Z" field are set to "1", status register protection circuit 261 causes the corresponding condition bit or bits in the status register to be protected from modification during execution of the instruction. Otherwise the status bits of status register 210 are set normally according to the resultant of arithmetic logic unit 230.

WEST

Generate Collection | Print

L23: Entry 1 of 2 File: USPT Apr 16, 1996

US-PAT-NO: 5509130
DOCUMENT-IDENTIFIER: US 5509130 A

TITLE: Method and apparatus for grouping multiple instructions, issuing grouped instructions simultaneously, and executing grouped instructions in a pipelined processor

DATE-ISSUED: April 16, 1996

INVENTOR-INFORMATION:

NAME	CITY	STATE	ZIP CODE	COUNTRY
Trauben; Richard D.	Morgan Hill	CA		
Nanda; Sunil	Los Altos	CA		

ASSIGNEE-INFORMATION:

NAME	CITY	STATE	ZIP CODE	COUNTRY	TYPE CODE
Sun Microsystems, Inc.	Mountain View	CA			02

APPL-NO: 8/ 355804 [PALM]
DATE FILED: December 14, 1994

PARENT-CASE:
This is a continuation of application Ser. No. 07/875,353 filed Apr. 29, 1992 now abandoned.

INT-CL: [6] G06 F 9/38

US-CL-ISSUED: 395/375; 395/800, 364/DIG.1
US-CL-CURRENT: 712/215; 712/206, 712/23

FIELD-OF-SEARCH: 395/375, 395/800

PRIOR-ART-DISCLOSED:

U.S. PATENT DOCUMENTS

Search Selected | Search ALL

	PAT-NO	ISSUE-DATE	PATENTEE-NAME	US-CL
☐	5129067	July 1992	Johnson	395/375
☐	5136697	August 1992	Johnson	395/375
☐	5226126	July 1993	McFarland et al.	395/375
☐	5226130	July 1993	Favor et al.	395/375
☐	5230068	July 1993	Van Dyke et al.	395/375

OTHER PUBLICATIONS

Machine organization of the IBM RISC System/6000 processor, Author: G. F. Grohoski; Publication: IBM Journal of Research & Development, vol. 34 No. 1, Jan. 1990.
A High Speed Superscalar PA-RISC 7100 Processor, Authors: Eric DeLano, Will Walker, Jeff Yetter, Mark Forsyth; Publication: Digest of Papers Compcon, Spring 1992.
The Motorola 88110 Superscalar RISC Microprocessor, Authors: Keith Diefendorff & Michael Allen; Publication: Digest of Papers Compcon, Spring 1992.
Abu-Nofal et al., "A Three-Million-Transistor Microprocessor," 1992 IEEE Solid-State Circuits Conference. Digest of Technical Papers, Feb. 19-21, 1992, pp. 108-109.
Kohn et al., "Introducing the Intel i860 64-Bit Microprocessor," IEEE Micro, Aug. 1989, pp. 15-30.
Lightner et al., "The Metaflow Lightning Chipset," Compcon Spring '91, IEEE, Conference Paper, Feb. 25-Mar. 1, 1991, pp. 13-18.
Popescu et al., "the Metaflow Architecture," IEEE Micro, Jun. 1991, pp. 10-13 and 63-73.
David J. Lilja, "Reducing the Branch Penalty in Piplined Processors," Computer, IEEE, Jul. 1988, pp. 47-55.
Smith et al., "Implementing Precise Interrups in Piplined Processors," IEEE Transactions on Computers, vol. 37, No. 5, May 1988, pp. 562-573.

ART-UNIT: 235
PRIMARY-EXAMINER: Treat; William M.
ATTY-AGENT-FIRM: Blakely Sokoloff Taylor & Zafman

ABSTRACT:

In a pipelined processor, an instruction queue and an instruction control unit is provided to group and issue m instructions simultaneously per clock cycle for execution. An integer and a floating point function unit capable of generating n.sub.1 and n.sub.2 integer and floating point results per clock cycle respectively, where n.sub.1 and n.sub.2 are sufficiently large to support m instructions being issued per clock cycle, is also provided to complement the instruction queue and instruction control unit. The pipeline stages are divided into integer and floating point pipeline stages where the early floating point stages overlap with the later integer pipeline stages. The instruction queue stores sequential instructions of a program and target instructions of a branch instruction of the program, fetched from the instruction cache. The instruction control unit decodes the instructions, detects operands cascading from instruction to instruction, group instructions into instruction groups of at most m instructions applying a number of exclusion rules, and issuing the grouped instructions simultaneously to the integer and/or floating point unit for execution. The exclusion rules reflect the resource characteristics and the particular implementation of the pipelined processor. The instruction control unit also tracks the history of the instruction groups and uses the history in conjunction with the exclusion rules in forming the instruction groups.

12 Claims, 12 Drawing figures

WEST

Generate Collection | Print

L23: Entry 1 of 2 File: USPT Apr 16, 1996

DOCUMENT-IDENTIFIER: US 5509130 A
TITLE: Method and apparatus for grouping multiple instructions, issuing grouped instructions simultaneously, and executing grouped instructions in a pipelined processor

Detailed Description Paragraph Right (97):
This exclusion rule terminates the current instruction group before an extended arithmetic instruction that follows an included instruction which sets the integer condition codes. This exclusion rule prevents the integer condition codes of the extended arithmetic instruction from being modified. For the exemplary pipelined processor instruction, multiple passes are made through the integer unit during the execution stage. The first pass is a conventional arithmetic instruction which modifies the condition codes for subsequent iterations. The subsequent passes use "extended" arithmetic instructions which use the integer condition codes as an additional source argument.

WEST

Generate Collection | Print

L22: Entry 22 of 129 File: USPT Jul 4, 2000

US-PAT-NO: 6085307
DOCUMENT-IDENTIFIER: US 6085307 A

TITLE: Multiple native instruction set master/slave processor arrangement and method thereof

DATE-ISSUED: July 4, 2000

INVENTOR-INFORMATION:

NAME	CITY	STATE	ZIP CODE	COUNTRY
Evoy; David Ross	Tempe	AZ		
Levy; Paul S.	Chandler	AZ		

ASSIGNEE-INFORMATION:

NAME	CITY	STATE	ZIP CODE	COUNTRY	TYPE CODE
VLSI Technology, Inc.	San Jose	CA			02

APPL-NO: 8/ 757151 [PALM]
DATE FILED: November 27, 1996

INT-CL: [7] G06 F 15/16

US-CL-ISSUED: 712/31; 712/1, 712/28, 712/29, 712/30, 712/33, 712/34, 712/200, 712/209, 395/500.01, 395/500.02, 395/500.4, 395/500.47, 395/500.49
US-CL-CURRENT: 712/31; 703/19, 703/26, 703/28, 712/1, 712/200, 712/209, 712/28, 712/29, 712/30, 712/33, 712/34, 716/1

FIELD-OF-SEARCH: 395/800.31, 395/800.41, 395/800.29, 395/800.3, 395/500.01, 395/500.02, 395/500.04, 395/500.47, 395/500.49, 712/1, 712/28, 712/29, 712/30, 712/31, 712/34, 712/200, 712/209

PRIOR-ART-DISCLOSED:

U.S. PATENT DOCUMENTS

Search Selected | Search ALL

	PAT-NO	ISSUE-DATE	PATENTEE-NAME	US-CL
☐	4876643	October 1989	McNeill et al.	364/200
☐	4888680	December 1989	Sander et al.	364/200
☐	5036453	July 1991	Renner et al.	364/200
☐	5073854	December 1991	Martin et al.	364/425
☐	5109329	April 1992	Strelioff	395/725
☐	5218711	June 1993	Yoshida	395/800
☐	5495588	February 1996	Gilbart et al.	395/375
☐	5590284	December 1996	Crosetto	395/200.05
☐	5778178	July 1998	Arunachalam	395/200.33

OTHER PUBLICATIONS

The ARM6 Family Bus Interface, Advanced RISC Machines Ltd. (1996).
The ARM Microprocessor Architecture, Advanced RISC Machines Ltd. (1996).
Wayner, "Sun Gambles On Java Chips", Byte, Nov. 1996.
Java Virtual Machine Architecture, Sun Microsystems (1996).

ART-UNIT: 273
PRIMARY-EXAMINER: Pan; Daniel H.
ASSISTANT-EXAMINER: Nguyen; Dzung C

ABSTRACT:

A multiple processor circuit arrangement utilizes a master processor which controls the operational state of a slave processor by programming internal control registers on the slave processor. In addition, a stack-based processor utilizes a stack cache for accelerating stack access operations and thereby accelerating the overall performance of the processor. When the stack-based processor is utilized as a slave processor in the aforementioned master/slave multi-processor computer system the slave processor is optimized to process platform-independent program code such as Java bytecodes, thereby permitting fast and efficient execution of both program code native to the master processor as well as platform-independent program code that is in effect native to the slave processor.

28 Claims, 14 Drawing figures

L22: Entry 22 of 129 File: USPT Jul 4, 2000

DOCUMENT-IDENTIFIER: US 6085307 A
TITLE: Multiple native instruction set master/slave processor arrangement and method thereof

Detailed Description Paragraph Right (22):
As shown in FIG. 2, blocks 56 and 58 are coupled to bus 21 through separate access ports 57, 59 because block 56 is capable of being "locked-out" from access by master processor 40. Block 56 may be locked-out by using three state buffers in access port 57 that are controlled or gated via the enable flag in block 58, or alternatively, block 56 may be considered locked-out from master processor 40 simply due to slave processor 50 asserting control over bus 21 through bus arbitrator 44. The direct access ports 57, 59 coupled to bus 21 are utilized by master processor 40 to control and monitor slave processor 50, and it is desirable to prevent master processor 40 from modifying the control registers and flags of slave processor 50 when the slave processor is running. However, to ensure the ability of the master processor to control slave processor 50, block 58 is not similarly locked-out. Other manners of accessing blocks 56, 58, including dedicated data or I/O ports or dedicated control lines, may also be used.

WEST

Generate Collection | Print

L22: Entry 128 of 129 File: USPT Jul 30, 1974

US-PAT-NO: 3827029
DOCUMENT-IDENTIFIER: US 3827029 A

TITLE: MEMORY AND PROGRAM PROTECTION SYSTEM FOR A DIGITAL COMPUTER SYSTEM

DATE-ISSUED: July 30, 1974

INVENTOR-INFORMATION:

NAME	CITY	STATE	ZIP CODE	COUNTRY
Schlotterer; John C.	Casselberry	FL		
Smith, Jr.; Lionel S.	San Jose	CA		

ASSIGNEE-INFORMATION:

NAME	CITY	STATE	ZIP CODE	COUNTRY	TYPE CODE
Westinghouse Electric Corporation	Pittsburgh	PA			02

APPL-NO: 5/ 292221 [PALM]
DATE FILED: September 25, 1972

INT-CL: [] G06f 11/00

US-CL-ISSUED: 340/172.5
US-CL-CURRENT: 711/163; 713/193

FIELD-OF-SEARCH: 340/172.5

PRIOR-ART-DISCLOSED:

U.S. PATENT DOCUMENTS

Search Selected | Search ALL

	PAT-NO	ISSUE-DATE	PATENTEE-NAME	US-CL
☐	3340539	September 1967	Sims, Jr.	340/172.5
☐	3573855	April 1971	Cragon et al.	340/172.5
☐	3585606	June 1971	Evans et al.	340/172.5
☐	3599159	December 1971	Creech et al.	340/172.5
☐	3671940	June 1972	Kronies et al.	340/172.5

ART-UNIT: 237
PRIMARY-EXAMINER: Zache; Raulfe B.
ASSISTANT-EXAMINER: Sachs; Michael

ATTY-AGENT-FIRM: Brodahl; R. G.

ABSTRACT:

A small size digital computer system is designed so that a hardware memory violation protect subsystem may be added to the computer system as a hardware option. The memory protect subsystem includes hardware which may operate in parallel with the digital computer system memory subsystem and which monitors each attempt to alter data within the memory subsystem. Any attempt to alter data within a protected region may be defeated. Following such an attempt, program execution is interrupted and program control is transferred to the computer system executive software. The computer system is also designed so that it may either modify or prevent the execution of certain instructions at times when the memory protect subsystem is in operation so as to defeat all attempts on the part of any software entity to destroy the integrity of the operating system.

10 Claims, 9 Drawing figures

L22: Entry 128 of 129 File: USPT Jul 30, 1974

DOCUMENT-IDENTIFIER: US 3827029 A
TITLE: MEMORY AND PROGRAM PROTECTION SYSTEM FOR A DIGITAL COMPUTER SYSTEM

Brief Summary Paragraph Right (17):
The only portions of the present invention which are incorporated into a basic minicomputer design are those portions which modify or prevent the normal operation of certain instructions when a particular processor flag is set. These portions may be incorporated into a typical minicomputer system with relatively little increase in the system cost. The basic minicomputer system may be then adapted to give memory and program protection in accordance with the invention by the simple insertion into the system of an additional card containing the protection subsystem hardware. Additional details relating to that computer system may be found in publication No. RF 2500-01, copies of which may be obtained from the same source.

WEST

Generate Collection | Print

L25: Entry 4 of 7 File: EPAB Jun 16, 1998

PUB-NO: US005768574A
DOCUMENT-IDENTIFIER: US 5768574 A
TITLE: Microprocessor using an instruction field to expand the condition flags and a computer system employing the microprocessor

PUBN-DATE: June 16, 1998

INVENTOR-INFORMATION:

NAME	COUNTRY
DUTTON, DREW J	US
CHRISTIE, DAVID S	US

ASSIGNEE-INFORMATION:

NAME	COUNTRY
ADVANCED MICRO DEVICES INC	US

APPL-NO: US91469897
APPL-DATE: August 19, 1997

PRIORITY-DATA: US91469897A (August 19, 1997)

INT-CL (IPC): G06 F 9/32
EUR-CL (EPC): G06F009/32; G06F009/318

ABSTRACT:

A microprocessor is provided which is configured to detect the presence of segment override prefixes in instruction code sequences being executed in flat memory mode, and to use the prefix value or the value stored in the associated segment register to selectively enable condition flag modification for instructions. An instruction which modifies the condition flags and a branch instruction intended to branch based on the condition flags set by the instruction may be separated by numerous instructions which do not modify the condition flags. When the branch instruction is decoded, the condition flags it depends on may already be available. In another embodiment of the present microprocessor, the segment register override bytes are used to select between multiple sets of condition flags. Multiple conditions may be retained by the microprocessor for later examination. Conditions which a program utilizes multiple times in a program may be maintained while other conditions may be generated and utilized.

WEST

End of Result Set

Generate Collection | Print

L26: Entry 1 of 1 File: DWPI Dec 7, 1999

DERWENT-ACC-NO: 2000-061861
DERWENT-WEEK: 200005

TITLE: X86 condition code modification method

INVENTOR: CHERNOFF, A; YATES, J S

PATENT-ASSIGNEE:

ASSIGNEE	CODE
DIGITAL EQUIP CORP	DIGI

PRIORITY-DATA: 1996US-0593286 (January 29, 1996)

PATENT-FAMILY:

PUB-NO	PUB-DATE	LANGUAGE	PAGES	MAIN-IPC
US 6000028 A	December 7, 1999		153	G06F009/45

APPLICATION-DATA:

PUB-NO	APPL-DATE	APPL-NO	DESCRIPTOR
US 6000028A	January 29, 1996	1996US-0593286	

INT-CL (IPC): G06 F 9/45

ABSTRACTED-PUB-NO: US 6000028A
BASIC-ABSTRACT:

NOVELTY - A table (139) includes entry for each X86 condition code modifying instruction. The table provides access to evaluation routines (140) which when executed modify one or all X86 condition code bits.

DETAILED DESCRIPTION - Another table includes entry for each X86 condition code bits that are associated with previously executed condition code modifying instruction and provides access to evaluation routines during which one or all X86 condition code bits are modified. The tables include pointers, respectively. An INDEPENDENT CLAIM is also included for X86 condition code modification apparatus.

USE - For modifying X86 condition code for execution of computer programs on nonnative computer system architecture.

ADVANTAGE - Needless evaluation of condition codes are avoided as condition code modification made by condition code instruction are never consumed by subsequent condition code modifying instruction, thus saving CPU processing time.

DESCRIPTION OF DRAWING(S) - The figure shows the block diagram of arrangement to determine evaluation routines for condition codes.

Table 139

Evaluation routines 140

CHOSEN-DRAWING: Dwg.14/71

TITLE-TERMS: CONDITION CODE MODIFIED METHOD

DERWENT-CLASS: T01

EPI-CODES: T01-D02; T01-E02X; T01-F05E; T01-S01C;

SECONDARY-ACC-NO:
Non-CPI Secondary Accession Numbers: N2000-048539

WEST

Generate Collection | Print

L37: Entry 4 of 30 File: USPT Dec 28, 1999

US-PAT-NO: 6009261
DOCUMENT-IDENTIFIER: US 6009261 A

TITLE: Preprocessing of stored target routines for emulating incompatible instructions on a target processor

DATE-ISSUED: December 28, 1999

INVENTOR-INFORMATION:

NAME	CITY	STATE	ZIP CODE	COUNTRY
Scalzi; Casper Anthony	Poughkeepsie	NY		
Schwarz; Eric Mark	Gardiner	NY		
Starke; William John	Austin	TX		
Urquhart; James Robert	Fishkill	NY		
Westcott; Douglas Wayne	Rhinebeck	NY		

ASSIGNEE-INFORMATION:

NAME	CITY	STATE	ZIP CODE	COUNTRY	TYPE CODE
International Business Machines Corporation	Armonk	NY			02

APPL-NO: 8/ 991714 [PALM]
DATE FILED: December 16, 1997

INT-CL: [6] G06 F 9/455

US-CL-ISSUED: 395/500.47; 395/500.48
US-CL-CURRENT: 703/26; 703/27

FIELD-OF-SEARCH: 257/51, 395/500.25, 395/500.35, 395/500.34, 395/500.21, 395/500.43, 395/500.47, 395/500.48, 395/500.49

PRIOR-ART-DISCLOSED:

U.S. PATENT DOCUMENTS

Search Selected | Search ALL

	PAT-NO	ISSUE-DATE	PATENTEE-NAME	US-CL
☐	4587612	May 1986	Fisk et al.	
☐	4851990	July 1989	Johnson et al.	
☐	5077657	December 1991	Cooper et al.	
☐	5081572	January 1992	Arnold	
☐	5333297	July 1994	Lemaire et al.	
☐	5471612	November 1995	Schlafly	707/104
☐	5488729	January 1996	Vegesna et al.	
☐	5574927	November 1996	Scantlin	395/500.44
☐	5751982	May 1998	Morley	712/209
☐	5857094	January 1999	Nemirovsky	395/500.49
☐	5896522	April 1999	Ward et al.	395/500.44
☐	5909567	June 1999	Novak et al.	712/208

OTHER PUBLICATIONS

U.S. application No. 08/864,585, Greenspan et al., filed May 28, 1997.
U.S. application No. 08/864,402, Greenspan et al., filed May 28, 1997.
Annexstein et al., Acheiving Multilanguage Behavior in Bit-Serial SIMD Architectures Via Emulation, Feb. 1990, pp. 186-195.
Hookway, Digital FX!32 Running 32-Bit X86 Applications on Alpha NT Mar. 1997, pp. 37-42.

ART-UNIT: 273
PRIMARY-EXAMINER: Teska; Kevin J.
ASSISTANT-EXAMINER: Fiul; Dan
ATTY-AGENT-FIRM: Ehrlich; Marc A. Goldman; Bernard M.

ABSTRACT:

Provides a program translation and execution method which stores target routines (for execution by a target processor) corresponding to incompatible instructions, interruptions and authorizations of an incompatible program written for execution on another computer system built to a computer architecture incompatible with the architecture of the target processor's computer system. The disclosed process allows the target processor to emulate incompatible acts expected in the operation of an incompatible program when the target processor itself is incapable of performing the emulated acts. Each of the instructions, interruptions and authorizations found in the incompatible programs has one or more corresponding target routines, any of which may need to be preprocessed before it can precisely emulate the execution results required by the incompatible architecture. Target routines (corresponding to the incompatible instruction instances in an incompatible program being emulated) are accessed, patched where necessary, and executed by a target processor to enable the target processor to precisely obtain the execution results of the emulated incompatible program. Before preprocessing, each target routine may not be able to provide identical execution results as required by the incompatible architecture, and the preprocessing may patch one or more of its target instructions to enable the target routine to perform the identical emulation execution of the corresponding incompatible instruction. The patching and other modifications to a target routine are done by one or more preprocessing instructions stored in the target routine.

47 Claims, 20 Drawing figures

WEST

Generate Collection | Print

L37: Entry 4 of 30 File: USPT Dec 28, 1999

DOCUMENT-IDENTIFIER: US 6009261 A
TITLE: Preprocessing of stored target routines for emulating incompatible instructions on a target processor

Brief Summary Paragraph Right (10):
The target instructions of a translation routine manage the unique architecture facilities of the incompatible program such as general purpose registers, control registers, instruction condition codes, program status words, etc. These are allocated to target machine registers or to target storage locations as appropriate, and processed in their mapped state by the target instructions of the translation routines.

Brief Summary Paragraph Right (11):
The target instructions also perform any special processes of the incompatible architecture such as address arithmetic, or providing special incompatible instruction condition code indications.

Current US Original Classification (1):
703/26

WEST

Generate Collection | Print

L37: Entry 8 of 30 File: USPT Jun 16, 1998

US-PAT-NO: 5768593
DOCUMENT-IDENTIFIER: US 5768593 A

TITLE: Dynamic cross-compilation system and method

DATE-ISSUED: June 16, 1998

INVENTOR-INFORMATION:

NAME	CITY	STATE	ZIP CODE	COUNTRY
Walters; Chad Perry	Redwood City	CA		
Brown; Jorg Anthony	Concord	CA		

ASSIGNEE-INFORMATION:

NAME	CITY	STATE	ZIP CODE	COUNTRY	TYPE CODE
Connectix Corporation	San Mateo	CA			02

APPL-NO: 8/ 620387 [PALM]
DATE FILED: March 22, 1996

INT-CL: [6] G06 F 9/30, G06 F 9/44

US-CL-ISSUED: 395/705; 395/707, 395/709, 395/500, 395/581
US-CL-CURRENT: 717/141; 703/26, 712/234

FIELD-OF-SEARCH: 395/700, 395/701-710, 395/712, 395/500, 395/581

PRIOR-ART-DISCLOSED:

U.S. PATENT DOCUMENTS

Search Selected | Search ALL

	PAT-NO	ISSUE-DATE	PATENTEE-NAME	US-CL
☐	4667290	May 1987	Goss et al.	395/707
☐	5167023	November 1992	De Nicolas et al.	395/527
☐	5179703	January 1993	Evans	395/703
☐	5204960	April 1993	Smith et al.	395/707
☐	5367683	November 1994	Brett	395/709
☐	5625822	April 1997	Brett	395/705
☐	5649203	July 1997	Sites	395/709
☐	5692196	November 1997	Unni et al.	395/705

ART-UNIT: 274
PRIMARY-EXAMINER: Voeltz; Emanuel Todd
ASSISTANT-EXAMINER: Dam; Tuan Q.
ATTY-AGENT-FIRM: Williams; Gary S. Flehr Hohbach Test Albritton & Herbert LLP

ABSTRACT:

In a computer system, a cross-compiler converts non-native code into native code immediately prior to execution of that code. The system also includes a code cache for storing cross-compiled code and a hash table for locating code blocks in the code cache. In a preferred embodiment, the system also includes an interpreter for emulating certain non-native instructions that are not converted into native code by the cross-compiler. While executing any non-native application, if the next instruction is not one of the predefined set of non-native instructions to be handled by interpretation or a special purpose procedure, then the next instruction is considered to be an "entry point" instruction, and the cross-compiler looks up the address of the entry point instruction in the hash table to see if a corresponding native code block is already stored in the code cache. If so, the native code block in the code cache is executed until an exit instruction in the native code block is encountered. Otherwise, the cross-compiler cross-compiles all code that is reachable from the entry point instruction during execution of the program without going outside the compilation window. During compilation the cross-compiler determines the non-native condition codes generated by a non-native instruction that will not be used by any successors of the non-native instruction. The native code instructions generated by the cross-compiler do not include instructions for processing non-native condition codes generated by the non-native instruction that will not be used by any successors of the qualifying non-native instruction.

12 Claims, 6 Drawing figures

WEST

Generate Collection | Print

L37: Entry 8 of 30 File: USPT Jun 16, 1998

DOCUMENT-IDENTIFIER: US 5768593 A
TITLE: Dynamic cross-compilation system and method

Abstract Paragraph Left (1):
In a computer system, a cross-compiler converts non-native code into native code immediately prior to execution of that code. The system also includes a code cache for storing cross-compiled code and a hash table for locating code blocks in the code cache. In a preferred embodiment, the system also includes an interpreter for emulating certain non-native instructions that are not converted into native code by the cross-compiler. While executing any non-native application, if the next instruction is not one of the predefined set of non-native instructions to be handled by interpretation or a special purpose procedure, then the next instruction is considered to be an "entry point" instruction, and the cross-compiler looks up the address of the entry point instruction in the hash table to see if a corresponding native code block is already stored in the code cache. If so, the native code block in the code cache is executed until an exit instruction in the native code block is encountered. Otherwise, the cross-compiler cross-compiles all code that is reachable from the entry point instruction during execution of the program without going outside the compilation window. During compilation the cross-compiler determines the non-native condition codes generated by a non-native instruction that will not be used by any successors of the non-native instruction. The native code instructions generated by the cross-compiler do not include instructions for processing non-native condition codes generated by the non-native instruction that will not be used by any successors of the qualifying non-native instruction.

Brief Summary Paragraph Right (11):
Another area in which existing cross-compilation systems generally fall short is their handling of condition codes. Condition codes are binary flag values generated by a data processor when executing various instructions, and that are used by various subsequent instructions to govern their execution. It is a well known fact that the condition codes used, and their exact definitions and usage, vary from computer platform to computer platform. As a result, often the majority of the native code generated by a cross-compiler is dedicated to keeping track of and using non-native condition codes (i.e., the condition codes associated with the computer platform for which the non-native application was written). However, the inventors of the present invention have determined that analysis of the program being cross-compiled can often substantially reduce the amount of native code required to track and use the non-native condition codes. It is therefore a goal of the present invention to determine the condition codes required by the instructions subsequent to a particular instruction, and to thereby avoid the generation of instructions for storing condition codes generated by that particular instruction but that are not required by any of the subsequent instructions.

Brief Summary Paragraph Right (12):
Since conditional branch instructions are often used at the end of execution loops in programs, conditional branch instructions are often executed large numbers of times. The inventors of the present invention have determined that optimization of the cross-compilation of such instructions is likely to have a disproportionately beneficial affect on the execution performance of cross-compiled programs. It is another goal of the present invention to minimize the native code instructions generated for non-native code conditional branch instructions by minimizing the number

of native code instructions used to handle non-native condition codes.

Brief Summary Paragraph Right (18):
During compilation of the qualifying code in a compilation window, the cross-compiler determines not only the non-native condition codes generated by each qualifying non-native instruction and the non-native condition codes needed to control the execution of the qualifying non-native instruction, but also determines all the non-native condition codes generated by the qualifying non-native instruction that will not be used by any successors of the qualifying non-native instruction. The native code instructions generated by the cross-compiler for a particular qualifying non-native instruction do not include any instructions for processing non-native condition codes generated by the qualifying non-native instruction that will not be used by any successors of the qualifying non-native instruction.

Brief Summary Paragraph Right (23):
In another aspect of the present invention, the cross-compiler of the preferred embodiment performs additional processing so as to minimize the native code instructions generated for non-native code conditional branch instructions by minimizing the number of native code instructions used to handle non-native condition codes. For instance, the cross-compiler determines whether a conditional branch instruction is the target of any other branch instruction(s) within the compilation window and whether it is immediately preceded by a comparison instruction. If the conditional branch instruction is the target of another branch instruction within the compilation window (or is an entry point instruction), and is immediately preceded by a comparison instruction, the cross-compiler converts the non-native comparison instruction into native instructions for generating and storing non-native condition codes, and converts the conditional branch instruction into a sequence of instructions for determining the status of the relevant non-native condition codes prior to performing a conditional branch. In a preferred embodiment the two non-native instructions are converted into eight native instructions.

Detailed Description Paragraph Right (20):
An information table 134 has a distinct entry 194 for every non-native instruction in the compilation window. Each information table entry has five components: a set of instruction flags 200, a set of "condition codes needed" (CCN) flags 202, a set of "condition codes modified" (CCM) flags 204, a set of "condition codes required" (CCR) flags 206, and a procedure address field 208. The instruction flags 200 include:

Detailed Description Paragraph Right (21):
Each of the sets of condition code flags contains one flag for each of the non-native condition codes associated with the non-native code that is being cross-compiled. In the preferred embodiment, there are five such condition codes, herein labeled X, N, Z, V and C. These five Motorola 68xxx condition codes do not have exact equivalents in PPC microprocessors and therefore have to be explicitly maintained by the cross-compiled code in order to exactly replicate the operation of the non-native code being cross-compiled.

Detailed Description Paragraph Right (22):
During compilation of the qualifying code in a compilation window, the cross-compiler determines not only the non-native condition codes generated by each qualifying non-native instruction and the non-native condition codes needed to control the execution of the qualifying non-native instruction, but also determines all the non-native condition codes generated by the qualifying non-native instruction will not be used by any successors of the qualifying non-native instruction.

Detailed Description Paragraph Right (25):
The portions of the information table entry generated by the decoder procedure for each qualifying instruction are: the instruction flags 200 (including setting the valid flag V to true), the CCN and CCM flags 202, 204, and the procedure address 208. The procedure address 208 stored in each information table entry 194 is: (A) the address of a code generation procedure 140 for the corresponding non-native code instruction, unless the instruction is a branch or jump instruction, in which case it is (B) the address of a condition code processing procedure 138 for the corresponding non-native code instruction. However, if the target of the jump or branch instruction is outside the compilation window, the "Branch" instruction flag is not set in the

corresponding information table entry 194, and the procedure address 208 stored in the corresponding information table entry 194 is the address of a code generation procedure 140 for that instruction. No special condition code processing is required for exit instructions, since the CCR flags 206 for exit instructions are always set to True.

Detailed Description Paragraph Right (28):
The second phase of the cross-compilation procedure is to generate the "condition code required" (CCR) flags 206 in the information table 134. The basic formula for computing the CCR flags 206 for any non-native instruction is:

Detailed Description Paragraph Right (29):
For "subroutine return" instructions and "jump" and "branch" instructions that branch outside the compilation window or branch to an unknown location, as well as any other instructions for which the successor instruction is not a qualifying instruction in the compilation window, the CCR flags are all set to true (224) because all of the condition code values may be needed by the successors to those instructions. These instructions that immediately precede exiting the compilation window are herein called "non-native code exit instructions."

Detailed Description Paragraph Right (31):
For branch and jump instructions (indicated in the information table by the "B" instruction flag being set to True), the procedure address in the corresponding information table entry is a condition code processing procedure. For unconditional branch and jump instructions that are not exit instructions, if the jump is a forward jump, the CCR is computed using the formula:

Detailed Description Paragraph Right (33):
Conditional branch instructions require additional processing because branch instructions have two successor instructions and the two execution paths may require the maintenance of different non-native condition codes. In particular, the CCR flag values for conditional branch instructions are computed using the following "CCR merger" formula: ##EQU2## where "s1" is the instruction index for a successor instruction other than the next instruction (i.e., the branch target instruction). If the branch target instruction corresponding to s1 is located before the conditional branch instruction (i.e., it is a backwards jump), the CCR(s1) value in the above formula is set to True before the value of the CCR(i) is computed. This is done because the CCR(s1) value has not yet been computed and setting it to True is the most conservative option available.

Detailed Description Paragraph Right (34):
For each distinct non-native branch and jump instruction in the non-native code language there is a corresponding distinct condition code processing procedure 138. Each such condition code processing procedure performs the corresponding CCR(i) computation step, as described above.

Detailed Description Paragraph Right (37):
Often the majority of the native code generated by a cross-compiler is dedicated to keeping track of and using non-native condition codes (i.e., the condition codes associated with the computer platform for which the non-native application was written). The CCR flags generated during the second phase of the compilation process are used to reduce the amount of native code required to track and use the non-native condition codes. In particular, during the third phase of the cross-compilation process, native code instructions are generated by each of the code generation procedures to store condition code values only for those non-native condition codes that are (A) generated by the current non-native instruction, and (B) that are required by successor instructions:

Detailed Description Paragraph Right (38):
As a result, the present invention avoids the generation of instructions for storing and manipulating non-native condition codes that are not used by any of the subsequent instructions.

Detailed Description Paragraph Right (40):
Since conditional branch instructions are often used at the end of execution loops in

programs, conditional branch instructions are often executed large numbers of times. The inventors have determined that optimization of the cross-compilation of such instructions is likely to have a disproportionately beneficial affect on the execution performance of cross-compiled programs. In particular, the code generation procedures for conditional branch instructions minimize the native code instructions generated by minimizing the number of native code instructions used to handle non-native condition codes.

Detailed Description Paragraph Right (41):
More particularly, the code generation procedure for each conditional branch instruction determines whether the conditional branch instruction for which code is being generated is the target of any other branch instruction(s) within the compilation window and whether it is immediately preceded by a comparison instruction. If the conditional branch instruction is the target of another branch instruction within the compilation window (or is an entry point instruction), and is immediately preceded by a comparison instruction, the code generation procedure converts the non-native comparison instruction into native instructions for generating and storing non-native condition codes, and converts the conditional branch instruction into a sequence of instructions for determining the status of the relevant non-native condition codes prior to performing a conditional branch. In a preferred embodiment the two non-native instructions:

Detailed Description Paragraph Right (43):
If execution of the non-native branch instruction is followed by execution of any instruction requiring the non-native condition codes, as indicated by the CCR flags for the conditional branch instruction, then two additional native instructions are generated to set and store those condition codes (i.e., the first two native instructions shown in the above listing of the eight standard native code instructions generated for a non-native conditional branch instruction). Otherwise, the native instructions for setting and storing the non-native condition codes are not generated.

Detailed Description Paragraph Right (44):
No instructions for reading stored non-native condition codes and branching based on the stored non-native condition codes (i.e., the third through seventh instructions in the above listing of the eight standard native code instructions generated for a non-native conditional branch instruction) are needed, because the branch instruction's operation is governed entirely by the immediately preceding comparison instruction. Thus, this optimization reduces the number of native instructions generated from eight to two, or eight to four, for executing the type of condition branch often found at the end of execution loops.

Detailed Description Paragraph Right (51):
The code optimizations implemented by any particular version of the cross-compiler will depend, in part, on the differences and similarities between the condition codes of the non-native and native code languages.

Detailed Description Paragraph Left (1):
where "&" indicates the logical AND operation, "i" is an instruction index indicating the instruction for which the CCR flags are being generated, and "i+1" is the instruction index for the next instruction to be executed immediately after the instruction for which the CCR flags are being generated. According to the above formula, the CCR flags for an instruction are the condition codes needed by the next instruction, as well as any condition codes required by the next instruction but excluding any condition code modified by that next instruction.

Detailed Description Paragraph Left (3):
where s1 is the target of the unconditional branch or jump instruction. This CCR computation formula, which is equivalent to the standard CCR computation formula with CCR(s1) set equal to True, uses the assumption that the successors to the branch or jump's target instruction require all condition codes.

Detailed Description Paragraph Type 1 (11):
a condition code requirement procedure 136, for determining which non-native condition codes must be maintained by the compiled native code; the condition code requirement

procedure 136 performs the second pass of the three pass compilation procedure;

Detailed Description Paragraph Type 1 (12):
a set of condition code merger procedures 138, that are utilized by the aforementioned condition code requirement procedure 136;

Detailed Description Paragraph Table (5):
tst.l d0 clears V flag, sets N if d0 < 0, sets Z flag if d0=0 blt.s @target branch to target if d0 is less than 0. Condition code basis for branch is: Not Z & ((V & Not N) or (Not V & N))

Detailed Description Paragraph Table (6):
addco. r3, r8, r0 clears C and V flags, sets Z and N flags based on value in r8 beq @x branch to x if equal to zero mfxer r5 condition code processing rlwinm r5, 45, 8, 24, 27 mtcrf 2, r5 crequ cr6+LT, cr0.sub.-- Lt, cr6.sub.-- GT blt- cr6, @target @x {successor instruction}

Current US Cross Reference Classification (1):
703/26

CLAIMS:

2. The method of claim 1, further including, prior to said cross-compiling step, storing in an information table for each of qualifying instructions data representing non-native condition codes used by said each qualifying instruction, non-native condition codes modified by said each qualifying instruction, and non-native condition codes required for use by successors of said each qualifying instruction; and

said cross-compiling step including generating for each said qualifying instruction a set of native code instructions to generate and store values for only those of said non-native condition codes that, in accordance with said data in said information table, are both modified by said qualifying instruction and are required for use by successors of said qualifying instruction.

3. The method of claim 2,

said cross-compiling step including determining for each said qualifying instruction composing a branch instruction that is not itself a target of any branch instruction among said qualifying instructions whether said branch instruction's immediate predecessor instruction always sets or always clears a non-native condition code utilized by said branch instruction, and when said determination is positive generating an optimized set of native code instructions for said preceding instruction and said branch instruction that do not process said non-native condition code that is always set or always cleared by said predecessor instruction.

8. The computer program product of claim 7,

said decoder procedure including instructions for (A3) storing in an information table for each of qualifying instructions data representing non-native condition codes used by said each qualifying instruction, non-native condition codes modified by said each qualifying instruction, and non-native condition codes required for use by successors of said each qualifying instruction; and

said cross-compiling procedure including instructions for generating for each said qualifying instruction a set of native code instructions to generate and store values for only those of said non-native condition codes that, in accordance with said data in said information table, are both modified by said qualifying instruction and are required for use by successors of said qualifying instruction.

9. The computer program product of claim 8,

said cross-compiling procedure including instructions for determining for each said qualifying instruction composing a branch instruction that is not itself a target of

any branch instruction among said qualifying instructions whether said branch instruction's immediate predecessor instruction always sets or always clears a non-native condition code utilized by said branch instruction, and when said determination is positive generating an optimized set of native code instructions for said preceding instruction and said branch instruction that do not process said non-native condition code that is always set or always cleared by said predecessor instruction.

WEST

Generate Collection | Print

L37: Entry 13 of 30 File: USPT Nov 19, 1996

US-PAT-NO: 5577233
DOCUMENT-IDENTIFIER: US 5577233 A

TITLE: Method and apparatus for direct conversion of programs in object code from between different hardware architecture computer systems

DATE-ISSUED: November 19, 1996

INVENTOR-INFORMATION:

NAME	CITY	STATE	ZIP CODE	COUNTRY
Goettelmann; John C.	Point Pleasant	NJ		
Macey; Christopher J.	Red Bank	NJ		

ASSIGNEE-INFORMATION:

NAME	CITY	STATE	ZIP CODE	COUNTRY	TYPE CODE
Lucent Technologies Inc.	Murray Hill	NJ			02

APPL-NO: 8/ 533243 [PALM]
DATE FILED: September 25, 1995

PARENT-CASE:
This is a continuation of application Ser. No. 616,507 filed Nov. 21, 1990 now abandoned, which is a continuation of application Ser. No. 280,774 filed Dec. 6, 1988 now U.S. Pat. No. 5,313,614.

INT-CL: [6] G06 F 15/38

US-CL-ISSUED: 395/500; 395/670
US-CL-CURRENT: 703/26; 709/100

FIELD-OF-SEARCH: 395/500, 395/700

PRIOR-ART-DISCLOSED:

U.S. PATENT DOCUMENTS

Search Selected | Search ALL

	PAT-NO	ISSUE-DATE	PATENTEE-NAME	US-CL
☐	4667290	May 1987	Goss et al.	395/700
☐	4791558	December 1988	Chaitin et al.	395/500
☐	4951195	August 1990	Fogg, Jr. et al.	395/375
☐	5142681	August 1992	Driscoll et al.	395/700

ART-UNIT: 235
PRIMARY-EXAMINER: Ellis; Richard L.
ATTY-AGENT-FIRM: Indyk; Eugene S. Slusky; Ronald D.

ABSTRACT:

Application programs compiled for a first, "source", computer are translated, from their object form, for execution on a section, "target", computer. The translated application programs are linked or otherwise bound with a translation of the source computer system software. The translated system software operates on the image of the source computer address space in the target computer exactly as it did in the source computer. The semantics of the source computer system softwae are thus preserved identically. In addition, a virtual hardware environment is provided in the target computer to manage events and to deal with differences in the address space layouts between the source and target computers.

10 Claims, 23 Drawing figures

L37: Entry 13 of 30 File: USPT Nov 19, 1996

DOCUMENT-IDENTIFIER: US 5577233 A
TITLE: Method and apparatus for direct conversion of programs in object code from between different hardware architecture computer systems

Brief Summary Paragraph Right (10):
As the term implies, the primary effect of any instruction is the fundamental purpose of the instruction. (For example, the primary effect of an ADD instruction is the addition of two quantities, while the primary effect of a JUMP insturciton is the changing of the flow of control.) By contrast, side effects include pieces of information resulting from the execution of an instruction which are maintained by the hardware for further reference by the executing program. (The term "side effect" as used herein is defined more rigorously hereinbelow.) Side effects include, for example, the setting of condition code registers. Inasmuch as the emulation software operates on an instruction-by-instruction basis and therefroe cannot determine when, if ever, such side effects will be referenced later in the execution, all side effects must be duplicated in the target machine. Because in the source machine these side effects are carried out by the processor hardware as an integral part of the instruction itself, they do not significantly affect performance in the source machine. However, differences in the processors usedx in the source and target machines are typically such that execution of the individual instructions of the software emulation on the target machien may result in side effects that are completely different from those of the source instructions executed on the source machine. Therefore, each side effect of each source machine instruction that does not have an equivalent on the target machine must be explicitly emulated on the target machine via a separate target machine instruction or instruction sequence. This results in a voluminous expansion of the program code.

Detailed Description Paragraph Right (86):
A side effect is any effect of an instruction that is a) other than its primary effect and b) is potentially useful at a later point in the program execution. A side effect may, for example, generate various pieces of "secondary" information. Thus, for example, side effects of arithmetic instructions often include the setting of condition code bits, the values of those bits being indicative of properties of either the arithmetic result (such as whether that result is zero or non-zero) or of the operation itself (such as whether the result overflowed). Alternatively, a side effect may relate not to the generation of secondary information, but rather to specifics of the semantics of an instruction. An example is whether the high-order half-word of a word-length register is or is not preserved when a half-word operation is performed on the register.

Detailed Description Paragraph Right (87):
It may be noted that a particular side effect may or may not have been intended by the processor design team. For example, it may turn out that, due to an error in the processor firmware, a particular condition code bit that is supposed to always be reset upon the execution of a particular instruction may be left un-reset when that instruction is executed with particular operand values.

Detailed Description Paragraph Right (88):
The manner in which side effects are used in a program are varied. For example, the value of a condition code bit may be tested in the very next program code instruction to change flow of control. Or the value of the overflow condition code bit may be

tested and, if it is found to have been set, an exception handler for overflow cases may be invoked. These are examples of routine uses of side effects which the processor design team provides because of their general utility. However, programmers often take advantage of particular side effects for other-than-originally-intended uses and, indeed, may take advantage of subtle side effects to this end.

Detailed Description Paragraph Right (89):
An example of the case where a particular non-subtle side-effect is used for other than its originally intended purpose is the setting of the carry condition code bit upon exit of a system call as a way of reporting back to the calling program that the system call failed. An example of a subtle side effect that may be used to advantage might be the fact that, when a half-word operation is performed on a word-length register, the high-order half-word of the register is preserved. This being so, a program may rely on this effect to pass data to subsequently executed portions of the program. If effect, the high-order half-word of the register functions as an extra half-word register. The mapping process must replicate all such side effects faithfully because the creator of the program being translated may have indeed relied on them in unforeseeable ways.

Detailed Description Paragraph Right (90):
The strategy for mapping of side effects is as follows: During the creation of the translator mapping phase, each instruction of the source machine instruction set is analyzed to appreciate both its primary effect and all of its side effects. Initially, the processor reference manual or other documentation can be used to gather information about the side effects. A skeleton intermediate language code sequence can then be developed for each source machine instruction. This sequence includes intermediate language instructions which perform the primary effect of the instruction and the known side effects. For example, the skeleton for an ADD instruction will include individual intermediate language statements which calculate and set the various condition code bits. (Although both the source and target machines may have nominally similar side effects, the detailed semantics of the side effects may be different. For example, the MOVE instruction on one machine may set the zero and negative condition code bits while the equivalent instruction on the other machine may not affect the condition code bits at all. Thus in preferred embodiments, the skeleton sequence will explicitly replicate the source machine side effects rather than relying on the corresponding target machine side effects.)

Detailed Description Paragraph Right (91):
Side effect peculiarities must also be taken into account. If, for example, a particular condition code bit--again through design or error--is not reset for particular operand values, this must be reproduced in the skeleton by providing code which, for example, tests those operand values and omits the setting of the condition code bit in the appropriate cases. In addition, any side effects which impact on the primary effect of the instruction being mapped must also be taken into account. Thus, for example, a half-word add must include the appropriate treatment of the high-order half-word of the word-length register, as in the example discussed above.

Detailed Description Paragraph Right (100):
In accordance with a feature of the invention, we have further realized that so-called data flow analysis techniques, heretofore developed and used in compiler technology, can be advantageously applied to the translation problem in order to perform the analysis and elimination of unnecessary instructions. Moreover, the data flow analysis is carried out in two parts, in accordance with a further feature of the invention, as discussed hereinbelow. Advantageously, the developed-for-compilers technique of definition-use chaining is illustratively employed to accomplish the data flow analysis. In particular, the object of the data flow analysis is to classify all mentions of variables in the intermediate language instructions as being either "definitions" or "uses". A "variable" is the name of a register (including condition code bits), memory location or stack location. A "definition" is an occurrence of a variable in an instruction in which the variable gets a new value. A "use" is any other occurrence of the variable, i.e., a place where the defined value is used. Each use, then, is a mention of a value created by some definition (or set of definitions) and, by examining the possible flow of control paths through a procedure, the set of definitions which could generate the value in a particular use are deduced. Thus looking ahead briefly to FIG. 21, the instruction "Z=10" is a definition of the

variable Z while the subsequent instruction "X=Z" is a use of variable Z. The process of associating definitions with uses is, in fact, the aforementioned definition-use chaining and the information thus generated is referred to herein as "the data flow results".

Current US Original Classification (1):
703/26

WEST

Generate Collection | Print

L37: Entry 21 of 30 File: USPT Dec 31, 1991

US-PAT-NO: 5077657
DOCUMENT-IDENTIFIER: US 5077657 A

TITLE: Emulator Assist unit which forms addresses of user instruction operands in response to emulator assist unit commands from host processor

DATE-ISSUED: December 31, 1991

INVENTOR-INFORMATION:

NAME	CITY	STATE	ZIP CODE	COUNTRY
Cooper; Thayne C.	West Valley City	UT		
Bell; Wayne D.	Bountiful	UT		

ASSIGNEE-INFORMATION:

NAME	CITY	STATE	ZIP CODE	COUNTRY	TYPE CODE
Unisys	Blue Bell	PA			02

APPL-NO: 7/ 367271 [PALM]
DATE FILED: June 15, 1989

INT-CL: [5] G06F 9/06, G06F 12/06, G06F 1/24

US-CL-ISSUED: 395/500; 364/238.2, 364/238.5, 364/241, 364/247.2, 364/254.4, 364/280.2, 364/280.8, 364/281.9, 364/284.4 , 364/927.96, 364/929, 364/929.4, 364/941.1, 364/957.2, 364/975.2, 364/DIG.1, 364/DIG.2
US-CL-CURRENT: 703/26

FIELD-OF-SEARCH: 364/2MSFile, 364/9MSFile

PRIOR-ART-DISCLOSED:

U.S. PATENT DOCUMENTS

Search Selected | Search ALL

	PAT-NO	ISSUE-DATE	PATENTEE-NAME	US-CL
☐	3374466	March 1968	Hanf et al.	364/200
☐	3544969	December 1970	Rokoczi et al.	364/200
☐	3646522	February 1972	Furman et al.	364/200
☐	3997895	December 1976	Cassonnet et al.	364/200
☐	4447876	May 1984	Moore	364/200
☐	4514803	April 1985	Agnew et al.	364/200
☐	4527234	July 1985	Bellay	364/200
☐	4587612	May 1986	Fisk et al.	364/200
☐	4591982	May 1986	Buonomo et al.	364/200
☐	4638423	January 1987	Ballard	364/200
☐	4695945	September 1987	Irwin	364/200
☐	4729094	September 1988	Zolnowsky et al.	364/200
☐	4763242	August 1988	Lee et al.	364/200
☐	4812975	March 1989	Adachi et al.	364/300
☐	4859995	August 1989	Hansen et al.	340/710
☐	4888680	December 1989	Sander et al.	364/200
☐	4920481	April 1990	Binkley et al.	364/200

ART-UNIT: 232
PRIMARY-EXAMINER: Lee; Thomas C.
ASSISTANT-EXAMINER: Kim; Ken S.
ATTY-AGENT-FIRM: Fassbender; Charles J. Starr; Mark T.

ABSTRACT:

An emulator is comprised of a host processor, an emulator, assist unit, and a memory which are closely coupled together over a co-processor bus. Stored in the memory is a user program which is a sequence of instructions from a user instruction set that is to be emulated, and a control program which is a mixture of host processor instructions and emulator assist unit instructions. In operation, the host processor reads and executes the hosts instructions, and it reads and passes the emulator assist unit instructions to the emulator assist unit for execution in that unit. By this means, the host processor and the emulator assist unit share the emulation tasks; and those tasks which are most difficult for the host are performed by the emulation assist unit. As one example the emulator assist unit has registers and controls which respond to the emulator assist unit instructions by examining the fields of the next user instruction that is to be emulated and by generating memory addresses of the operands which that next user instruction operates on; while the host uses those addresses to read the operands from memory and perform operations on the operands as specified by the user instruction.

10 Claims, 10 Drawing figures

L37: Entry 21 of 30 File: USPT Dec 31, 1991

DOCUMENT-IDENTIFIER: US 5077657 A
TITLE: Emulator Assist unit which forms addresses of user instruction operands in response to emulator assist unit commands from host processor

Detailed Description Paragraph Right (4):
For comparison purposes, FIG. 2A illustrates the format of the Intel instructions, whereas FIG. 2B illustrates the format of the Motorola instructions. Inspection of FIG. 2A shows that the Intel 80.times.86 instructions have a 2-bit "MOD" field, a 3-bit "REG" field, and a 3-bit "R/M" field which together specify the location of one or two operands. By comparison, FIG. 2B shows that such fields don,t even exist in the Motorola instructions. Further, the Intel 80.times.86 instructions use the content of several registers which are called AX, BX, CX, DX, BP, SP, DI, SI, ES, CS, SS, DS, FS and GS; and these registers don't exist in the Motorola 68030 microprocessor. Also, several of the Intel 80.times.86 instructions cause certain condition codes to be set to indicate the instruction results, and these same condition codes are not in the 68030 microprocessor.

Detailed Description Paragraph Right (5):
Due to the above differences and others between the Intel 80.times.86 instructions and the Motorola 68030 instructions, it is very cumbersome to emulate the Intel instructions by means of a 68030 software program. If 68030 instructions are used to examine all the different fields of the the Intel 80.times.86 instructions, generate the various operand addresses, fetch the operands and perform byte swapping, perform the called-for operation, and set condition codes, then the resulting overall execution time is very slow.

Detailed Description Paragraph Right (24):
E-FLAG. This register holds the condition codes which are set/reset to indicate the results of an Intel 80.times.86 instruction. The various bits of this register indicate the following conditions.

Detailed Description Paragraph Right (25):
Also included in the emulator assist unit 13 are five adders 13a-13e, a shifter 13f, effective address control logic 13g, an instruction decoder 13h, and condition code logic 13i. These components 13a-13i are interconnected to the emulator assist unit register as FIG. 3 illustrates.

Detailed Description Paragraph Right (28):
Instruction decoder 13h receives as an input emulator assist unit instructions, and in response it generates control signals INSTCNTL, CSCNTL, DCNTL, and IPCNTL. The INSTCNTL signals load respective portions of the 80.times.86 instruction into the E-PRE, E-OP, and E-EA registers; and the CSCNTL signals pass the content of the CS register into adder 13a; the DCNTL signals load the displacement and data portions of an 80.times.86 instruction into the E-DISP and E-DATA registers; and the IPCNTL signal passes the content of the IP register into the adder 13b. In like fashion, the condition code logic examines the content of registers E-RES, E-SRC1, and E-SRC2; and, in response to an emulator assist unit instruction, it generates control signals CCCNTL which set the condition codes in the E-FLAG register. Here again, these tasks cannot be efficiently performed by the host processor.

Detailed Description Paragraph Right (40):

Next, at time t.sub.10, the host processor reads another control program instruction which it then passes to the emulator assist unit 13. In response, the condition code logic 13i uses the content of registers E-RES, E-SRC1, and E-SRC2 to set the indicators in the E-FLAG register such that they reflect the result of the Intel ADD instruction.

Current US Original Classification (1):
703/26

WEST Search History

DATE: Friday, May 17, 2002

Set Name side by side	Query	Hit Count	Set Name result set
DB=USPT,PGPB; PLUR=NO; OP=ADJ			
L42	L38 same instruction$1	57	L42
L41	L38 same (emulat$3 or native or risc)	5	L41
L40	L38 with (emulat$3 or native or risc)	1	L40
L39	L38 same emulat$3	2	L39
L38	enabl$3 with (modification or alteration or chang$3 or modify$3 or alter$4) with (flag$1 or (condition code$1))	457	L38
L37	L32 and condition code$1	30	L37
L36	L35 and condition code$1	0	L36
L35	l32 and l15	0	L35
L34	l32 and l20	7	L34
L33	L32 and l9	1	L33
L32	((703/26)!.CCLS.)	219	L32
L31	5630137.uref.	2	L31
L30	6000028.uref.	3	L30
L29	5768574.uref.	4	L29
DB=JPAB,EPAB,DWPI,TDBD; PLUR=NO; OP=ADJ			
L28	L27 with (emulat$3 or native or risc)	4	L28
L27	(condition code$1 or flag$1 or status register) with (modif$7 or destruction or destroy$3 or overwrit$3 or over-writ$3)	461	L27
L26	condition code modification	1	L26
L25	flag modification	7	L25
DB=USPT,PGPB; PLUR=NO; OP=ADJ			
L24	l22 with (emulat$3 or native or risc)	0	L24
L23	L20 with condition code$1	2	L23

L22	L20 with flag$1	129	L22
L21	L20 with status register	5	L21
L20	prevent$3 with (modif$7 or destruction or destroy$3 or overwrit$3 or over-writ$3)	35806	L20
L19	L18 not l17	1	L19
L18	l15 with (modif$7 or destruction or destroy$3 or overwrit$3 or over-writ$3)	47	L18
L17	l15 with (modif$7 or (destruction or destroy$3 or overwrit$3 or over-writ$3) over-writ$3)	46	L17
L16	L15 with emulat$3	0	L16
L15	protect$3 with status register	171	L15
L14	l9 with instruction$1	37	L14
L13	l9 with risc	0	L13
L12	L9 with native	2	L12
L11	l9 with emulat$3	4	L11
L10	L9 with (destruction or destroy$3 or overwrit$3 or over-writ$3)	55	L10
L9	protect$3 with (flag$1)	1055	L9
L8	protect$3 with (condition code$1)	31	L8
L7	l1 not l4	35	L7
L6	l1 not l3	53	L6
L5	l1L4	6	L5
L4	l1 and emulat$3	18	L4
L3	(risc and cisc) and L1	0	L3
L2	native and L1	0	L2
L1	(over-writ$3 or overwrit$3) with flag$1 with prevent$3	53	L1

END OF SEARCH HISTORY

WEST

Generate Collection | Print

L42: Entry 16 of 57 File: USPT Jun 16, 1998

US-PAT-NO: 5768574
DOCUMENT-IDENTIFIER: US 5768574 A

TITLE: Microprocessor using an instruction field to expand the condition flags and a computer system employing the microprocessor

DATE-ISSUED: June 16, 1998

INVENTOR-INFORMATION:

NAME	CITY	STATE	ZIP CODE	COUNTRY
Dutton; Drew J.	Austin	TX		
Christie; David S.	Austin	TX		

ASSIGNEE-INFORMATION:

NAME	CITY	STATE	ZIP CODE	COUNTRY	TYPE CODE
Advanced Micro Devices, Inc.	Sunnyvale	CA			02

APPL-NO: 8/ 914698 [PALM]
DATE FILED: August 19, 1997

PARENT-CASE:
This application is a continuation of application Ser. No. 08/481,704, filed Jun. 7, 1995 now abandoned.

INT-CL: [6] G06 F 9/32

US-CL-ISSUED: 395/567; 711/206, 711/209
US-CL-CURRENT: 712/226; 711/206, 711/209

FIELD-OF-SEARCH: 395/567, 711/206, 711/209

PRIOR-ART-DISCLOSED:

U.S. PATENT DOCUMENTS

Search Selected | Search ALL

	PAT-NO	ISSUE-DATE	PATENTEE-NAME	US-CL
☐	4044338	August 1977	Wolf	
☐	4050094	September 1977	Bourke	395/416
☐	4109311	August 1978	Blum et al.	395/567
☐	4385352	May 1983	Bienvenu	
☐	4453212	June 1984	Gaither et al.	
☐	4807115	February 1989	Torng	
☐	4835734	May 1989	Kodaira et al.	395/419
☐	4858105	August 1989	Kuriyama et al.	
☐	4926322	May 1990	Stimac et al.	
☐	4972338	November 1990	Crawford et al.	
☐	5109334	April 1992	Kamuro	
☐	5125087	June 1992	Randell	
☐	5226126	July 1993	McFarland et al.	
☐	5226130	July 1993	Favor et al.	
☐	5226132	July 1993	Yamamoto et al.	
☐	5274834	December 1993	Kardach et al.	
☐	5293592	March 1994	Fu et al.	
☐	5303358	April 1994	Baum	395/567
☐	5321836	June 1994	Crawford et al.	
☐	5375213	December 1994	Arai	
☐	5438668	August 1995	Coon et al.	
☐	5471593	November 1995	Branigin	395/375
☐	5560032	September 1996	Nguyen et al.	395/800
☐	5561784	October 1996	Chen et al.	395/484

FOREIGN PATENT DOCUMENTS

FOREIGN-PAT-NO	PUBN-DATE	COUNTRY	US-CL
A 0 067 667	December 1982	EPX	
0259095	March 1988	EPX	
A 0 369 396	May 1990	EPX	
0381471	August 1990	EPX	
A 0 425 410	May 1991	EPX	
0459232	December 1991	EPX	
A 0 467 152	January 1992	EPX	
2263987	August 1993	GBX	
2263985	August 1993	GBX	
2281422	March 1995	GBX	

OTHER PUBLICATIONS

Intel, "Chapter 2: Microprocessor Architecture Overview," pp. 2-1 through 2-4.
Michael Slater, "AMD's K5 Designed to Outrun Pentium," Microprocessor Report, vol. 8, No. 14, Oct. 24, 1994, 7 pages.
Sebastian Rupley and John Clyman, "P6: The Next Step?," PC Magazine, Sep. 12, 1995, 16 pages.
Tom R. Halfhill, "AMD K6 Takes On Intel P6," Byte, Jan. 1996, 4 pages.
IEEE Micro, vol. 13, No. 5, Oct. 1, 1993, pp. 24-36, XP000397921, Makoto Awaga et al.: "The VP 64-Bit Vector Coprocessor: A New Implementation of High-Performance Numerical Computation".

ART-UNIT: 235
PRIMARY-EXAMINER: Lall; Parshotam S.
ASSISTANT-EXAMINER: Patel; Gautam R.
ATTY-AGENT-FIRM: Conley, Rose & Tayon Kivlin; B. Noel

ABSTRACT:

A microprocessor is provided which is configured to detect the presence of segment override prefixes in instruction code sequences being executed in flat memory mode, and to use the prefix value or the value stored in the associated segment register to selectively enable condition flag modification for instructions. An instruction which modifies the condition flags and a branch instruction intended to branch based on the condition flags set by the instruction may be separated by numerous instructions which do not modify the condition flags. When the branch instruction is decoded, the condition flags it depends on may already be available. In another embodiment of the present microprocessor, the segment register override bytes are used to select between multiple sets of condition flags. Multiple conditions may be retained by the microprocessor for later examination. Conditions which a program utilizes multiple times in a program may be maintained while other conditions may be generated and utilized.

14 Claims, 6 Drawing figures

WEST

Generate Collection | Print

L42: Entry 16 of 57 File: USPT Jun 16, 1998

DOCUMENT-IDENTIFIER: US 5768574 A
TITLE: Microprocessor using an instruction field to expand the condition flags and a computer system employing the microprocessor

Abstract Paragraph Left (1):
A microprocessor is provided which is configured to detect the presence of segment override prefixes in instruction code sequences being executed in flat memory mode, and to use the prefix value or the value stored in the associated segment register to selectively enable condition flag modification for instructions. An instruction which modifies the condition flags and a branch instruction intended to branch based on the condition flags set by the instruction may be separated by numerous instructions which do not modify the condition flags. When the branch instruction is decoded, the condition flags it depends on may already be available. In another embodiment of the present microprocessor, the segment register override bytes are used to select between multiple sets of condition flags. Multiple conditions may be retained by the microprocessor for later examination. Conditions which a program utilizes multiple times in a program may be maintained while other conditions may be generated and utilized.

Brief Summary Paragraph Right (9):
The problems outlined above are in large part solved by a microprocessor according to the present invention. The microprocessor is configured to detect the presence of segment override prefixes in instruction code sequences being executed in flat memory mode, and to use the prefix value or the value stored in the associated segment register to selectively enable or disable condition flag modification for instructions. Advantageously, an instruction which modifies the condition flags and a branch instruction intended to branch based on the condition flags set by the instruction may be separated by numerous instructions which do not modify the condition flags. When the branch instruction is decoded, the condition flags it depends on may already be available. Therefore, the branch instruction may be executed more quickly than if it were placed directly subsequent to the instruction upon which it depends.

WEST

Generate Collection | Print

L47: Entry 15 of 37 File: USPT Jul 20, 1999

US-PAT-NO: 5925124
DOCUMENT-IDENTIFIER: US 5925124 A

TITLE: Dynamic conversion between different instruction codes by recombination of instruction elements

DATE-ISSUED: July 20, 1999

INVENTOR-INFORMATION:

NAME	CITY	STATE	ZIP CODE	COUNTRY
Hilgendorf; Rolf	Boblingen			DEX
Schwermer; Hartmut	Stuttgart			DEX
Soell; Werner	Schonaich			DEX

ASSIGNEE-INFORMATION:

NAME	CITY	STATE	ZIP CODE	COUNTRY	TYPE CODE
International Business Machines Corporation	Armonk	NY			02

APPL-NO: 8/ 810880 [PALM]
DATE FILED: March 5, 1997

FOREIGN-APPL-PRIORITY-DATA:

COUNTRY	APPL-NO	APPL-DATE
EP	97103233	February 27, 1997

INT-CL: [6] G06 F 9/30

US-CL-ISSUED: 712/227; 712/209, 712/226
US-CL-CURRENT: 712/227; 712/209, 712/226

FIELD-OF-SEARCH: 395/385, 395/568, 395/567, 395/384, 395/386, 395/387, 395/388, 395/389

PRIOR-ART-DISCLOSED:

U.S. PATENT DOCUMENTS

Search Selected | Search ALL

	PAT-NO	ISSUE-DATE	PATENTEE-NAME	US-CL
☐	4236204	November 1980	Groves	395/567
☐	4415969	November 1983	Bayliss et al.	395/387
☐	4791559	December 1988	Byers	395/389
☐	4839797	June 1989	Katori et al.	395/386
☐	5115500	May 1992	Larsen	395/385
☐	5455955	October 1995	Kida et al.	395/384
☐	5568646	October 1996	Jaggar	395/358
☐	5638525	June 1997	Hammond et al.	395/385

ART-UNIT: 274
PRIMARY-EXAMINER: Ellis; Richard L.
ATTY-AGENT-FIRM: Ehrlich; Marc A.

ABSTRACT:

The invention provides an apparatus and a method for converting instructions of a code A to instructions of a code B. Said conversion is performed by obtaining rearrangement information, which corresponds to the instruction that is to be converted, from a table. Said rearrangement information is then used to rearrange the instruction elements of the initial instruction, in order to generate instructions of code B, which functionally corresponds to said initial instruction. Said rearrangement can be performed by multiplexing means, which use said instruction elements of the initial code A instruction as input, and which select one of said instruction elements, or the content of another register, and forward this selected data to the instruction that is to be generated. Said rearrangement information is directly used to control the selection performed by said multiplexers.

20 Claims, 5 Drawing figures

L47: Entry 15 of 37 File: USPT Jul 20, 1999

DOCUMENT-IDENTIFIER: US 5925124 A
TITLE: Dynamic conversion between different instruction codes by recombination of instruction elements

Detailed Description Paragraph Right (20):
Logical source registers have to be specified by 5 bits (318, 319), whereby condition code used as a source operand (317) requires two bits, in our implementation. Five select bits are necessary to route the correct instruction part to the "immediate" data field (316). In order to route the correct code A instruction part to the target data field of instruction 2, five select bits are necessary (315). To indicate whether said internal instruction 2 depends on the actual condition code, 2 bits have to be provided (317). To indicate whether said instruction 2 generates new condition code, which means that the actual condition code is modified, two bits are required (314).

Detailed Description Paragraph Right (22):
The actual condition code can also be used as a source operand, for example in a "branch on condition" operation. In these cases, it is indicated in data field 405 that the actual condition code is to be used as one of the instruction's sources.

Detailed Description Paragraph Right (23):
It also has to be specified, in the instruction, where result data is to be written to. Data field 409 contains the name of the logical register that is to be used as a target register. Additionally, an instruction might modify the actual condition code and thus produce a new condition code value. In this case, the new condition code is to be treated as a target operand, and this is indicated in data field 408.

Detailed Description Paragraph Right (35):
Source data field 405 and target data field 408 refer to the handling of condition code. In case the internal instruction 400 depends on the actual condition code, "CC" is forwarded by multiplexer 514 to data field 405. This indicates that the actual condition code is to be used as a source operand. In case the internal instruction does not use condition code as a source operand, "CN" is written to data field 405.

Detailed Description Paragraph Right (36):
The target data field 408 is handled accordingly. In case the internal instruction 400 modifies the actual condition code, the new condition code is considered as a target operand, and "CC" is written to data field 408. Otherwise, "CX" is stored to data field 408.

Current US Original Classification (1):
712/227

WEST

Generate Collection | Print

L49: Entry 2 of 9 File: USPT Jan 9, 2001

US-PAT-NO: 6173394
DOCUMENT-IDENTIFIER: US 6173394 B1

TITLE: Instruction having bit field designating status bits protected from modification corresponding to arithmetic logic unit result

DATE-ISSUED: January 9, 2001

INVENTOR-INFORMATION:

NAME	CITY	STATE	ZIP CODE	COUNTRY
Guttag; Karl M.	Missouri City	TX		
Poland; Sydney W.	Katy	TX		
Balmer; Keith	Bedford			GBX

ASSIGNEE-INFORMATION:

NAME	CITY	STATE	ZIP CODE	COUNTRY	TYPE CODE
Texas Instruments Incorporated	Dallas	TX			02

APPL-NO: 9/ 372470 [PALM]
DATE FILED: August 11, 1999

PARENT-CASE:
This application is a division of Application No. 08/160,118 filed Nov. 30, 1993, now U.S. Pat. No. 6,058,473.

INT-CL: [7] G06 F 9/308, G06 F 9/318

US-CL-ISSUED: 712/226; 712/221, 712/224, 708/525
US-CL-CURRENT: 712/226; 708/525, 712/221, 712/224

FIELD-OF-SEARCH: 712/220, 712/221, 712/223, 712/224, 712/225, 712/226, 708/525

PRIOR-ART-DISCLOSED:

U.S. PATENT DOCUMENTS

Search Selected | Search ALL

	PAT-NO	ISSUE-DATE	PATENTEE-NAME	US-CL
☐	4785393	November 1988	Chu et al.	712/221

ART-UNIT: 278
PRIMARY-EXAMINER: Vu; Viet D.
ATTY-AGENT-FIRM: Marshall, Jr.; Robert D. Brady, III; W. James Telecky, Jr.; Frederick

J.

ABSTRACT:

A data processing apparatus includes plural data registers, an arithmetic logic unit and a status register. The status register stores a plurality of different types of status bits. These status bits could be a negative status bit, a carry status bit, an overflow status bit and a zero status bit. These status bits are normally set dependent upon the condition of the result generated by the current arithmetic logic unit operation. A status bit protect instruction type permits selection of status bits protected from modification corresponding to the current arithmetic logic unit result. This status bit protect instruction preferably includes individual protect bit corresponding to each status bit. If a protect bit has a first digital state, then the corresponding status bit may be modified corresponding to the current arithmetic logic unit result. If the protect bit has a second opposite digital state, then the corresponding status bit is protected from modification according to the arithmetic logic unit results.

19 Claims, 71 Drawing figures

L49: Entry 2 of 9 File: USPT Jan 9, 2001

DOCUMENT-IDENTIFIER: US 6173394 B1
TITLE: Instruction having bit field designating status bits protected from modification corresponding to arithmetic logic unit result

Detailed Description Paragraph Right (121):
FIG. 19 illustrates in block diagram form the construction of an exemplary bit circuit 400 of arithmetic logic unit 230. Arithmetic logic unit 230 preferably operates on data words of 32 bits and thus consists of 32 bit circuits 400 in parallel. Each bit circuit 400 of arithmetic logic unit 230 receives: the corresponding bits of the three inputs A.sub.i, B.sub.i and C.sub.i ; a zero carry-in signal designated c.sub.in0 from the previous bit circuit 400; a one carry-in signal designated c.sub.in1 from the previous bit circuit 400; an arithmetic enable signal A.sub.en ; an inverse kill signal K.sub.i-1 from the previous bit circuit; a carry sense select signal for selection of carry-in signal c.sub.in0 or c.sub.in1 ; and eight inverse function signals F7-F0. The carry-in signals c.sub.in0 and c.sub.in1 for the first bit (bit 0) are identical and are generated by a special circuit that will be described below. Note that the input signals A.sub.i, B.sub.i and C.sub.i are formed for each bit of arithmetic logic unit 230 and may differ. The arithmetic enable signal A.sub.en and the inverted function signals F7-F0are the same for all of the 32 bit circuits 400. Each bit circuit 400 of arithmetic logic unit 230 generates: a corresponding one bit resultant S.sub.i ; an early zero signal Z.sub.i ; a zero carry-out signal designated c.sub.out0 that forms the zero carry-in signal c.sub.in0 for the next bit circuit; a one carry-out signal designated c.sub.out1 that forms the one carry-in signal c.sub.in1 for the next bit circuit; and an inverse kill signal K.sub.i that forms the inverse kill signal K.sub.i-1 for the next bit circuit. A selected one of the zero carry-out signal c.sub.out0 or the one carry-out signal c.sub.out1 of the last bit in the 32 bit arithmetic logic unit 230 is stored in status register 210, unless the "C" bit is protected from change for that instruction. In addition during multiple arithmetic the instruction may specify that carry-out signals from separate arithmetic logic unit sections be stored in multiple flags register 211. In this event the selected zero carry-out signal c.sub.out0 or the one carry-out signal c.sub.out1 will be stored in multiple flags register 211.

Detailed Description Paragraph Right (405):
The "N C V Z" field (bits 28-25) indicates which bits of the status are protected from alteration during execution of the instruction. The conditions of the status register are: N negative; C carry; V overflow; and Z zero. If one or more of these bits are set to "1", the corresponding condition bit or bits in the status register are protected from modification during execution of the instruction. Otherwise the status bits of status register 210 are set normally according to the resultant of arithmetic logic unit 230.

Detailed Description Paragraph Right (520):
The subtraction of instruction 1a effectively compares the numbers Data1 and Data2. If Data1Data2, then neither of these bits are set. This example illustrates another use of the write priority rules of Table 51. The unconditional address unit register move of Dummy to Dummy, protects Dummy from change while permitting status register 210 to be set based upon the arithmetic logic unit result. The register to register move aborts storing the arithmetic logic unit result. If instruction 1a sets the negative "N" status bit, the instruction 2b sets Data1 equal to zero. Otherwise instruction 2a sets Data1 equal to Data2. Thus instruction 2 performs the operation "if Data1

Current US Original Classification (1):
712/226

WEST Search History

DATE: Friday, May 17, 2002

Set Name side by side	Query	Hit Count	Set Name result set
DB=USPT,PGPB; PLUR=NO; OP=ADJ			
L51	L50 and (l9 or l20 or l15)	12	L51
L50	((712/229)!.CCLS.)	173	L50
L49	L48 and (l9 or l20 or l15)	9	L49
L48	((712/226)!.CCLS.)	202	L48
L47	L43 and condition code$1	37	L47
L46	L43 and l15	0	L46
L45	L43 and l20	9	L45
L44	L43 and l9	1	L44
L43	((712/227)!.CCLS.)	281	L43
L42	L38 same instruction$1	57	L42
L41	L38 same (emulat$3 or native or risc)	5	L41
L40	L38 with (emulat$3 or native or risc)	1	L40
L39	L38 same emulat$3	2	L39
L38	enabl$3 with (modification or alteration or chang$3 or modify$3 or alter$4) with (flag$1 or (condition code$1))	457	L38
L37	L32 and condition code$1	30	L37
L36	L35 and condition code$1	0	L36
L35	l32 and l15	0	L35
L34	l32 and l20	7	L34
L33	L32 and l9	1	L33
L32	((703/26)!.CCLS.)	219	L32
L31	5630137.uref.	2	L31
L30	6000028.uref.	3	L30
L29	5768574.uref.	4	L29
DB=JPAB,EPAB,DWPI,TDBD; PLUR=NO; OP=ADJ			

L28	L27 with (emulat$3 or native or risc)	4	L28
L27	(condition code$1 or flag$1 or status register) with (modif$7 or destruction or destroy$3 or overwrit$3 or over-writ$3)	461	L27
L26	condition code modification	1	L26
L25	flag modification	7	L25
DB=USPT,PGPB; PLUR=NO; OP=ADJ			
L24	l22 with (emulat$3 or native or risc)	0	L24
L23	L20 with condition code$1	2	L23
L22	L20 with flag$1	129	L22
L21	L20 with status register	5	L21
L20	prevent$3 with (modif$7 or destruction or destroy$3 or overwrit$3 or over-writ$3)	35806	L20
L19	L18 not l17	1	L19
L18	l15 with (modif$7 or destruction or destroy$3 or overwrit$3 or over-writ$3)	47	L18
L17	l15 with (modif$7 or (destruction or destroy$3 or overwrit$3 or over-writ$3) over-writ$3)	46	L17
L16	L15 with emulat$3	0	L16
L15	protect$3 with status register	171	L15
L14	l9 with instruction$1	37	L14
L13	l9 with risc	0	L13
L12	L9 with native	2	L12
L11	l9 with emulat$3	4	L11
L10	L9 with (destruction or destroy$3 or overwrit$3 or over-writ$3)	55	L10
L9	protect$3 with (flag$1)	1055	L9
L8	protect$3 with (condition code$1)	31	L8
L7	l1 not l4	35	L7
L6	l1 not l3	53	L6
L5	l1L4	6	L5
L4	l1 and emulat$3	18	L4
L3	(risc and cisc) and L1	0	L3
L2	native and L1	0	L2

L1	(over-writ$3 or overwrit$3) with flag$1 with prevent$3	53	L1

END OF SEARCH HISTORY

WEST Search History

DATE: Friday, May 17, 2002

Set Name side by side	Query	Hit Count	Set Name result set
DB=USPT,PGPB; PLUR=NO; OP=ADJ			
L51	L50 and (l9 or l20 or l15)	12	L51
L50	((712/229)!.CCLS.)	173	L50
L49	L48 and (l9 or l20 or l15)	9	L49
L48	((712/226)!.CCLS.)	202	L48
L47	L43 and condition code$1	37	L47
L46	L43 and l15	0	L46
L45	L43 and l20	9	L45
L44	L43 and l9	1	L44
L43	((712/227)!.CCLS.)	281	L43
L42	L38 same instruction$1	57	L42
L41	L38 same (emulat$3 or native or risc)	5	L41
L40	L38 with (emulat$3 or native or risc)	1	L40
L39	L38 same emulat$3	2	L39
L38	enabl$3 with (modification or alteration or chang$3 or modify$3 or alter$4) with (flag$1 or (condition code$1))	457	L38
L37	L32 and condition code$1	30	L37
L36	L35 and condition code$1	0	L36
L35	l32 and l15	0	L35
L34	l32 and l20	7	L34
L33	L32 and l9	1	L33
L32	((703/26)!.CCLS.)	219	L32
L31	5630137.uref.	2	L31
L30	6000028.uref.	3	L30
L29	5768574.uref.	4	L29
DB=JPAB,EPAB,DWPI,TDBD; PLUR=NO; OP=ADJ			

L28	L27 with (emulat$3 or native or risc)	4	L28
L27	(condition code$1 or flag$1 or status register) with (modif$7 or destruction or destroy$3 or overwrit$3 or over-writ$3)	461	L27
L26	condition code modification	1	L26
L25	flag modification	7	L25
DB=USPT,PGPB; PLUR=NO; OP=ADJ			
L24	l22 with (emulat$3 or native or risc)	0	L24
L23	L20 with condition code$1	2	L23
L22	L20 with flag$1	129	L22
L21	L20 with status register	5	L21
L20	prevent$3 with (modif$7 or destruction or destroy$3 or overwrit$3 or over-writ$3)	35806	L20
L19	L18 not l17	1	L19
L18	l15 with (modif$7 or destruction or destroy$3 or overwrit$3 or over-writ$3)	47	L18
L17	l15 with (modif$7 or (destruction or destroy$3 or overwrit$3 or over-writ$3) over-writ$3)	46	L17
L16	L15 with emulat$3	0	L16
L15	protect$3 with status register	171	L15
L14	l9 with instruction$1	37	L14
L13	l9 with risc	0	L13
L12	L9 with native	2	L12
L11	l9 with emulat$3	4	L11
L10	L9 with (destruction or destroy$3 or overwrit$3 or over-writ$3)	55	L10
L9	protect$3 with (flag$1)	1055	L9
L8	protect$3 with (condition code$1)	31	L8
L7	l1 not l4	35	L7
L6	l1 not l3	53	L6
L5	l1L4	6	L5
L4	l1 and emulat$3	18	L4
L3	(risc and cisc) and L1	0	L3
L2	native and L1	0	L2

L1	(over-writ$3 or overwrit$3) with flag$1 with prevent$3	53	L1

END OF SEARCH HISTORY

WEST Search History

DATE: Friday, May 17, 2002

Set Name side by side	Query	Hit Count	Set Name result set
	DB=USPT,PGPB; PLUR=NO; OP=ADJ		
L54	L53 and risc	1	L54
L53	6173394.pn.	1	L53
L52	6058473.pn.	1	L52
L51	L50 and (l9 or l20 or l15)	12	L51
L50	((712/229)!.CCLS.)	173	L50
L49	L48 and (l9 or l20 or l15)	9	L49
L48	((712/226)!.CCLS.)	202	L48
L47	L43 and condition code$1	37	L47
L46	L43 and l15	0	L46
L45	L43 and l20	9	L45
L44	L43 and l9	1	L44
L43	((712/227)!.CCLS.)	281	L43
L42	L38 same instruction$1	57	L42
L41	L38 same (emulat$3 or native or risc)	5	L41
L40	L38 with (emulat$3 or native or risc)	1	L40
L39	L38 same emulat$3	2	L39
L38	enabl$3 with (modification or alteration or chang$3 or modify$3 or alter$4) with (flag$1 or (condition code$1))	457	L38
L37	L32 and condition code$1	30	L37
L36	L35 and condition code$1	0	L36
L35	l32 and l15	0	L35
L34	l32 and l20	7	L34
L33	L32 and l9	1	L33
L32	((703/26)!.CCLS.)	219	L32
L31	5630137.uref.	2	L31

L30	6000028.uref.	3	L30
L29	5768574.uref.	4	L29
DB=JPAB,EPAB,DWPI,TDBD; PLUR=NO; OP=ADJ			
L28	L27 with (emulat$3 or native or risc)	4	L28
L27	(condition code$1 or flag$1 or status register) with (modif$7 or destruction or destroy$3 or overwrit$3 or over-writ$3)	461	L27
L26	condition code modification	1	L26
L25	flag modification	7	L25
DB=USPT,PGPB; PLUR=NO; OP=ADJ			
L24	l22 with (emulat$3 or native or risc)	0	L24
L23	L20 with condition code$1	2	L23
L22	L20 with flag$1	129	L22
L21	L20 with status register	5	L21
L20	prevent$3 with (modif$7 or destruction or destroy$3 or overwrit$3 or over-writ$3)	35806	L20
L19	L18 not l17	1	L19
L18	l15 with (modif$7 or destruction or destroy$3 or overwrit$3 or over-writ$3)	47	L18
L17	l15 with (modif$7 or (destruction or destroy$3 or overwrit$3 or over-writ$3) over-writ$3)	46	L17
L16	L15 with emulat$3	0	L16
L15	protect$3 with status register	171	L15
L14	l9 with instruction$1	37	L14
L13	l9 with risc	0	L13
L12	L9 with native	2	L12
L11	l9 with emulat$3	4	L11
L10	L9 with (destruction or destroy$3 or overwrit$3 or over-writ$3)	55	L10
L9	protect$3 with (flag$1)	1055	L9
L8	protect$3 with (condition code$1)	31	L8
L7	l1 not l4	35	L7
L6	l1 not l3	53	L6
L5	l1L4	6	L5
L4	l1 and emulat$3	18	L4

L3	(risc and cisc) and L1	0	L3
L2	native and L1	0	L2
L1	(over-writ$3 or overwrit$3) with flag$1 with prevent$3	53	L1

END OF SEARCH HISTORY

WEST

Generate Collection | Print

Search Results - Record(s) 1 through 1 of 1 returned.

□ 1. Document ID: US 6173394 B1

L54: Entry 1 of 1 File: USPT Jan 9, 2001

DOCUMENT-IDENTIFIER: US 6173394 B1
TITLE: Instruction having bit field designating status bits protected from modification corresponding to arithmetic logic unit result

Detailed Description Paragraph Right (15):
Master processor 60 preferably performs the major control functions for multiprocessor integrated circuit 100. Master processor 60 is preferably a 32 bit reduced instruction set computer (RISC) processor including a hardware floating point calculation unit. According to the RISC architecture, all accesses to memory are performed with load and store instructions and most integer and logical operations are performed on registers in a single cycle. The floating point calculation unit, however, will generally take several cycles to perform operations when employing the same register file as used by the integer and logical unit. A register score board ensures that correct register access sequences are maintained. The RISC architecture is suitable for control functions in image processing. The floating point calculation unit permits rapid computation of image rotation functions, which may be important to image processing.

Detailed Description Paragraph Right (28):
Data unit 110 performs all of the logical and arithmetic operations during the execute pipeline stage. All logical and arithmetic operations and all data movements to or from memory occur during the execute pipeline stage. The global data port and the local data port complete any memory accesses, which are begun during the address pipeline stage, during the execute pipeline stage. The global data port and the local data port perform all data alignment needed by memory stores, and any data extraction and sign extension needed by memory loads. If the program counter is specified as a data destination during any operation of the execute pipeline stage, then a delay of two instructions is experienced before any branch takes effect. The pipelined operation requires this delay, since the next two instructions following such a branch instruction have already been fetched. According to the practice in RISC processors, other useful instructions may be placed in the two delay slot positions.

Generate Collection | Print

Term	Documents
RISC.USPT,PGPB.	7436
(53 AND RISC).USPT,PGPB.	1
(L53 AND RISC).USPT,PGPB.	1

Display Format: - Change Format

Previous Page Next Page

L9: Entry 1 of 63 File: PGPB Aug 8, 2002

DOCUMENT-IDENTIFIER: US 20020108103 A1
TITLE: Intercalling between native and non-native instruction sets

Summary of Invention Paragraph (21):
[0020] Examples of known systems for translation between instruction sets and other background information may be found in the following: U.S. Pat. Nos. 5,805,895; 3,955,180; 5,970,242; 5,619,665; 5,826,089; 5,925,123; 5,875,336; 5,937,193; 5,953,520; 6,021,469; 5,568,646; 5,758,115; and 5,367,685; IBM Technical Disclosure Bulletin, March 1988, pp308-309, "System/370 Emulator Assist Processor For a Reduced Instruction Set Computer"; IBM Technical Disclosure Bulletin, July 1986, pp548-549, "Full Function Series/1 Instruction Set Emulator"; IBM Technical Disclosure Bulletin, March 1994, pp605-606, "Real-Time CISC Architecture HW Emulator On A RISC Processor"; IBM Technical Disclosure Bulletin, March 1998, p272, "Performance Improvement Using An EMULATION Control Block"; IBM Technical Disclosure Bulletin, January 1995, pp537-540, "Fast Instruction Decode For Code Emulation on Reduced Instruction Set Computer/Cycles Systems"; IBM Technical Disclosure Bulletin, February 1993, pp231-234, "High Performance Dual Architecture Processor"; IBM Technical Disclosure Bulletin, August 1989, pp40-43, "System/370 I/O Channel Program Channel Command Word Prefetch"; IBM Technical Disclosure Bulletin, June 1985, pp305-306, "Fully Microcode-Controlled Emulation Architecture"; IBM Technical Disclosure Bulletin, March 1972, pp3074-3076, "Op Code and Status Handling For Emulation"; IBM Technical Disclosure Bulletin, August 1982, pp954-956, "On-Chip Microcoding of a Microprocessor With Most Frequently Used Instructions of Large System and Primitives Suitable for Coding Remaining Instructions"; IBM Technical Disclosure Bulletin, April 1983, pp5576-5577, "Emulation Instruction"; the book ARM System Architecture by S Furber; the book Computer Architecture: A Quantitative Approach by Hennessy and Patterson; and the book The Java Virtual Machine Specification by Tim Lindholm and Frank Yellin 1.sup.st and 2.sup.nd editions.

WEST

Generate Collection Print

L10: Entry 10 of 18 File: USPT Jul 20, 1999

US-PAT-NO: 5926642
DOCUMENT-IDENTIFIER: US 5926642 A

TITLE: RISC86 instruction set

DATE-ISSUED: July 20, 1999

INVENTOR-INFORMATION:

NAME	CITY	STATE	ZIP CODE	COUNTRY
Favor; John G.	San Jose	CA		

ASSIGNEE-INFORMATION:

NAME	CITY	STATE	ZIP CODE	COUNTRY	TYPE CODE
Advanced Micro Devices, Inc.	Sunnyvale	CA			02

APPL-NO: 08/ 649983 [PALM]
DATE FILED: May 16, 1996

PARENT-CASE:
RELATED APPLICATIONS This application claims benefit of U.S. Provisional Application No. 60/005,069, filed Oct. 6, 1995 and U.S. Provisional Application No. 60/005,021, filed Oct. 10, 1995, and is a continuation-in-part of application Ser. No. 08/592,151, filed on Jan. 26, 1996.

INT-CL: [06] G06 F 15/76

US-CL-ISSUED: 395/800.01; 395/200.33
US-CL-CURRENT: 712/1; 709/203

FIELD-OF-SEARCH: 395/800.01, 395/800.23, 395/800.32, 395/800.41, 395/200.3, 395/200.33, 395/200.53, 395/670, 395/376, 395/384-386

PRIOR-ART-DISCLOSED:

U.S. PATENT DOCUMENTS

Search Selected Search ALL

	PAT-NO	ISSUE-DATE	PATENTEE-NAME	US-CL
☐	4641262	February 1987	Bryan et al.	345/168
☐	4868765	September 1989	Diefendorff	345/345
☐	4932048	June 1990	Kenmochi et al.	379/67
☐	4941089	July 1990	Fischer	395/200.61
☐	5018146	May 1991	Sexton	371/37.07
☐	5101341	March 1992	Circello et al.	395/375
☐	5131086	July 1992	Circello et al.	395/375
☐	5155843	October 1992	Stamm et al.	395/182.03
☐	5185868	February 1993	Tran	395/375
☐	5201056	April 1993	Daniel et al.	395/800
☐	5222244	June 1993	Carbine et al.	395/800
☐	5233694	August 1993	Hotta et al.	395/391
☐	5233696	August 1993	Suzuki	395/375
☐	5287490	February 1994	Sites	395/500
☐	5301342	April 1994	Scott	711/111
☐	5333277	July 1994	Searls	395/281
☐	5394524	February 1995	DiNicola et al.	345/506
☐	5412766	May 1995	Pietras et al.	345/431
☐	5438668	August 1995	Coon et al.	395/375
☐	5440619	August 1995	Cann	379/93.11
☐	5488710	January 1996	Sato et al.	395/452
☐	5495419	February 1996	Rostoker et al.	364/468.2
☐	5504689	April 1996	Fiebrich et al.	702/122
☐	5513330	April 1996	Stiles	395/375
☐	5568646	October 1996	Jaggar	395/385
☐	5598546	January 1997	Blomgren	395/385
☐	5619665	April 1997	Emma	395/384
☐	5638525	June 1997	Hammond et al.	395/385
☐	5646676	July 1997	Dewkett et al.	348/7
☐	5664215	September 1997	Burgess et al.	395/800.23
☐	5758114	May 1998	Johnson et al.	395/380
☐	5758141	May 1998	Kahle et al.	395/570

FOREIGN PATENT DOCUMENTS

FOREIGN-PAT-NO	PUBN-DATE	COUNTRY	US-CL
0 380 854 A3	August 1990	EP	
0 454 984 A2	June 1991	EP	
0 506 972 A1	July 1992	EP	
0 498 654 A3	December 1992	EP	
0 651 320	May 1995	EP	
2 263 987	August 1993	GB	
2 263 985	August 1993	GB	
WO 93/01546	January 1993	WO	
WO 93/20507	October 1993	WO	

OTHER PUBLICATIONS

Diefendorff, K. et al: "The PowerPC User Instruction Set Architecture", Oct. 1, 1994, pp. 30-41, IEEE Micro, XP000476678.
John Crawford: "Architecture of the Intel 80386", Sep. 6, 1986, pp. 155-160, IEEE Int. Conf. on Computer Design: VLSI in Comnputers, XP000212275.
Steven, G.B., et al.: "Iharp: A Multiple Instruction Issue Processor" pp. 439-449, IEEE Proceedings E. Computers & Digital Techniques, XP000319892.
Steve MacGeady: "Inside Intel's i960CA superscalar processor", Jul., 1990, pp. 385-396, Microprocessors and Microsystems, vol. 14, No. 6 XP000151094.

ART-UNIT: 273

PRIMARY-EXAMINER: Shah; Alpesh M.

ABSTRACT:

An internal RISC-type instruction structure furnishes a fixed bit-length template including a plurality of defined bit fields for a plurality of operation (Op) formats. One format includes an instruction-type bit field, two source-operand bit fields and one destination-operand bit field for designating a register-to-register operation. Another format is a load-store format that includes an instruction-type bit field, an identifier of a source or destination register for the respective load or store operation, and bit fields for specifying the segment, base and index parameters of an address.

38 Claims, 16 Drawing figures

L10: Entry 10 of 18 File: USPT Jul 20, 1999

DOCUMENT-IDENTIFIER: US 5926642 A
TITLE: RISC86 instruction set

Detailed Description Text (139):
The extension field (EXT) 614 is used, in combination with bit <0>of the TYPE field 612, for MOVcc operations to specify a 5-bit condition code. The extension field (EXT) 614 is also used for RDxxx/WRxxx operations to specify a 4-bit special register number. The set status (SS) field 624, in combination with the EXT field 614, is used to designate the status flags that are affected by an operation. For Ops in which the set status (SS) field 624 is set to 1, indicating that this Op does modify flags, the extension field (EXT) 614 specifies four status modification bits designating the groups of flags that are modified by the Op. For RDSEG Ops, the EXT field 614 specifies a 4-bit segment (selector) register. For a WRFLG conditional Op, the special register encoding matches the desired StatMod value for when the SS field is set. The set status (SS) field 624 and the EXT field 614 are RegOp fields.

Detailed Description Text (141):
In the definitions, the character ".about." indicates a logical NOT operation, "." designates a logical AND operation, "+" specifies a logical OR operation, and " " designates a logical XOR operation. OF, SF, ZF, AF, PF, and CF are standard x86 status bits. EZF and ECF are an emulation zero flag and an emulation carry flag, respectively, that emulation code uses in sequences implementing x86 instructions when the architectural zero flag ZF and carry flag CF are not changed. IP, DTF and SSTF are signals indicating an interrupt is pending, a debuf trap flag and a single step trap flag respectively.

Detailed Description Text (143):
The EXT field 614 is used to update condition flags including six flags corresponding to x86 flags and two emulation flags. The eight flags are divided into four groups, using one status modification bit per group of flags. The EXT field 614 defines updating of the various condition code flags substantially independent of the TYPE 612 specification, so that functionally related flags are controlled and updated as a group. Updating of related flags as a group advantageously conserves control logic. The EXT field 614 defines a set of bits which determine the flags to be updated for a particular instruction. Decoupling of condition code handling from operation type, using the independent TYPE 612 and set status (SS) field 624, allows some operations to be defined which do not update the flags. Accordingly, for those circumstances in which updating of condition flags is not necessary, it is highly advantageous to disable flag updating to avoid unnecessary dependency on previous flag values.

Detailed Description Text (144):
The RU1-only field 616 is indicative of Ops that are issued to the first register unit 244 and not to the second register unit 246 so that the R1 field 616 is a bit for hard-encoding an execution unit specification. Thus, the RU1-only field 616 indicates that a particular operation is only executable on a specific execution unit, generally because only the specific execution unit incorporates a function implemented on that unit. The set status (SS) field 624 modifies status flags according to EXT field 614 settings. The I field 626 specifies whether the source 2 operand is immediate or a general register. The IMM8/SRC2 field 628 specifies a 5-bit general register if the I field 626 is zero. The IMM8/SRC2 field 628 specifies a signed immediate value which is extended to the size of operation specified by the DSz field size if the I field 626 is one.

Detailed Description Text (173):
For example, in one processor 120 embodiment, an emulation carry flag is defined in addition to the conventional carry flag and associated with the emulation environment. Thus, a conventional add instruction is defined that sets the carry flag and changes the permanent architectural state of the processor 120. An emulation add operation is defined that sets the emulation carry flag and not the conventional carry flag. Various instructions are defined to branch on the basis of the emulation carry flag while other instructions branch on the basis of the conventional carry flag. Thus, the processor 120 is concurrently operational in both a conventional microarchitectural state and in an emulation microarchitectural state. This operation in an emulation environment allows the processor 120 to execute an emulation microsequence and to not change the visible microarchitectural state.

Detailed Description Text (250):
StatMod[3:0] Status group marks indicating which groups of status bits the Op modifies. Bits [3,2,1,0] correspond to {EZF,ECF}, OF, {SF,ZF,AF,PF}, and CF respectively. StatMod is always all zeroes for non-RegOps and is cleared during abort cycles.

Detailed Description Text (361):
There is an additional input from the RUX unit (RUX.sub.-- NoStatMod) which indicates that the operation being executed there does not modify status flags. A cycle-delayed version, called "NoStatMod," is useful in several situations. The following equations describe this logic.

US Reference Patent Number (25):
5568646

US Reference Patent Number (27):
5619665

WEST

Generate Collection Print

L10: Entry 3 of 18 File: USPT Jan 1, 2002

US-PAT-NO: 6336178
DOCUMENT-IDENTIFIER: US 6336178 B1

TITLE: RISC86 instruction set

DATE-ISSUED: January 1, 2002

INVENTOR-INFORMATION:

NAME	CITY	STATE	ZIP CODE	COUNTRY
Favor; John G.	San Jose	CA		

ASSIGNEE-INFORMATION:

NAME	CITY	STATE	ZIP CODE	COUNTRY	TYPE CODE
Advanced Micro Devices, Inc.	Sunnyvale	CA			02

APPL-NO: 09/ 152043 [PALM]
DATE FILED: September 11, 1998

PARENT-CASE:
RELATED APPLICATIONS This application is a division of application Ser. No. 08/649,983, filed May 16, 1996, now U.S. Pat. No. 5,926,642, which is a continuation-in-part of application Ser. No. 08/592,151, filed on Jan. 26, 1996 now abandoned, and claims benefit of U.S. Provisional Application No. 60/005,069, filed Oct. 6, 1995 and U.S. Provisional Application No. 60/005,021, filed Oct. 10, 1995.

INT-CL: [07] G06 F 15/00

US-CL-ISSUED: 712/23; 712/213, 712/209
US-CL-CURRENT: 712/23; 712/209, 712/213

FIELD-OF-SEARCH: 712/23, 712/205, 712/208, 712/209, 712/210, 712/215, 712/212, 712/21T, 712/213, 717/5

PRIOR-ART-DISCLOSED:

U.S. PATENT DOCUMENTS

Search Selected Search ALL

	PAT-NO	ISSUE-DATE	PATENTEE-NAME	US-CL
☐	4641262	February 1987	Bryan et al.	364/900
☐	4868765	September 1989	Diefendorff	364/521
☐	4932048	June 1990	Kenmochi et al.	379/67
☐	4941089	July 1990	Fischer	364/200
☐	5018146	May 1991	Sexton	371/37.1
☐	5101341	March 1992	Circello et al.	395/375
☐	5131086	July 1992	Circello et al.	395/375
☐	5155843	October 1992	Stamm et al.	395/575
☐	5185868	February 1993	Tran	395/375
☐	5201056	April 1993	Daniel et al.	395/800
☐	5222244	June 1993	Carbine et al.	395/800
☐	5233694	August 1993	Hotta et al.	395/375
☐	5233696	August 1993	Suzuki	395/375
☐	5287490	February 1994	Sites	395/500
☐	5301342	April 1994	Scott	395/800
☐	5333277	July 1994	Searls	395/325
☐	5394524	February 1995	DiNicola et al.	395/163
☐	5412466	May 1995	Pietras et al.	395/131
☐	5438668	August 1995	Coon et al.	395/375
☐	5440619	August 1995	Cann	379/97
☐	5488710	January 1996	Sato et al.	395/452
☐	5495419	February 1996	Rostoker et al.	364/468
☐	5504689	April 1996	Fiebrich et al.	364/481
☐	5513330	April 1996	Stiles	395/375
☐	5568646	October 1996	Jaggar	395/800
☐	5598546	January 1997	Blomgren	395/385
☐	5619665	April 1997	Emma	395/384
☐	5638525	June 1997	Hammond et al.	395/385
☐	5646676	July 1997	Dewkett et al.	348/7
☐	5664215	September 1997	Burgess et al.	395/800.23
☐	5758114	May 1998	Johnson et al.	395/380
☐	5758141	May 1998	Kahle et al.	395/570

FOREIGN PATENT DOCUMENTS

FOREIGN-PAT-NO	PUBN-DATE	COUNTRY	US-CL
0 380 854	August 1990	EP	
0 454 984	November 1991	EP	
0 498 654	August 1992	EP	
0 506 972	October 1992	EP	
0 651 320	May 1995	EP	
2 263 985	August 1993	GB	
2 263 987	August 1993	GB	
WO 93/01546	January 1993	WO	
WO 93/20507	October 1993	WO	

OTHER PUBLICATIONS

Steve McGeady, "Inside Intel's i960CA Superscalar Processor," Jul. 1990, pp. 385-396,Microprocessors and Microsystems, vol. 14, No. 6, XP000151094.
Keith Diefendorff and Ed Silha, "The PowerPC User Instruction Set Architecture," Oct. 14, 1994, pp. 30-41, IEEE Micro, Los Alamitos, CA, XP000476678.
John Crawford, "Architecture of the Intel 80386," Oct. 6-9, 1986, pp. 155-160, IEEE Int. Conf. on Computer Design: VLSI in Computers, XP000212275.
G. B. steven et al., "iHARP: A Multiple Instruction Issue Processor," Sep. 1992, pp. 439-449, IEEE Proceedings-E, vol. 139, No. 5, XP000319892.

ART-UNIT: 2154

PRIMARY-EXAMINER: Donaghue; Larry D.

ABSTRACT:

An internal RISC-type instruction structure furnishes a fixed bit-length template including a plurality of defined bit fields for a plurality of operation (Op) formats. One format includes an instruction-type bit field, two source-operand bit fields and one destination-operand bit field for designating a register-to-register operation. Another format is a load-store format that includes an instruction-type bit field, an identifier of a source or destination register for the respective load or store operation, and bit fields for specifying the segment, base and index parameters of an address.

48 Claims, 16 Drawing figures

L10: Entry 3 of 18 File: USPT Jan 1, 2002

DOCUMENT-IDENTIFIER: US 6336178 B1
TITLE: RISC86 instruction set

Detailed Description Text (132):
The extension field (EXT) 614 is used, in combination with bit <0> of the TYPE field 612, for MOVcc operations to specify a 5-bit condition code. The extension field (EXT) 614 is also used for RDxxx/WRxxx operations to specify a 4-bit special register number. The set status (SS) field 624, in combination with the EXT field 614, is used to designate the status flags that are affected by an operation. For Ops in which the set status (SS) field 624 is set to 1, indicating that this Op does modify flags, the extension field (EXT) 614 specifies four status modification bits designating the groups of flags that are modified by the Op. For RDSEG Ops, the EXT field 614 specifies a 4-bit segment (selector) register. For a WRFLG conditional Op, the special register encoding matches the desired StatMod value for when the SS field is set. The set status (SS) field 624 and the EXT field 614 are RegOp fields.

Detailed Description Text (135):
In the definitions, the character ".about." indicates a logical NOT operation, ".multidot." designates a logical AND operation, "+" specifies a logical OR operation, and " " designates a logical XOR operation. OF, SF, ZF, AF, PF, and CF are standard x86 status bits. EZF and ECF are an emulation zero flag and an emulation carry flag, respectively, that emulation code uses in sequences implementing x86 instructions when the architectural zero flag ZF and carry flag CF are not changed. IP, DTF and SSTF are signals indicating an interrupt is pending, a debuf trap flag and a single step trap flag respectively.

Detailed Description Text (137):
The EXT field 614 is used to update condition flags including six flags corresponding to x86 flags and two emulation flags. The eight flags are divided into four groups, using one status modification bit per group of flags. The EXT field 614 defines updating of the various condition code flags substantially independent of the TYPE 612 specification, so that functionally related flags are controlled and updated as a group. Updating of related flags as a group advantageously conserves control logic. The EXT field 614 defines a set of bits which determine the flags to be updated for a particular instruction. Decoupling of condition code handling from operation type, using the independent TYPE 612 and set status (SS) field 624, allows some operations to be defined which do not update the flags. Accordingly, for those circumstances in which updating of condition flags is not necessary, it is highly advantageous to disable flag updating to avoid unnecessary dependency on previous flag values.

Detailed Description Text (138):
The RU1-only field 616 is indicative of Ops that are issued to the first register unit 244 and not to the second register unit 246 so that the R1 field 616 is a bit for hard-encoding an execution unit specification. Thus, the RU1-only field 616 indicates that a particular operation is only executable on a specific execution unit, generally because only the specific execution unit incorporates a function implemented on that unit. The set status (SS) field 624 modifies status flags according to EXT field 614 settings. The I field 626 specifies whether the source 2 operand is immediate or a general register. The IMM8/SRC2 field 628 specifies a 5-bit general register if the I field 626 is zero. The IMM8/SRC2 field 628 specifies a signed immediate value which is extended to the size of operation specified by the DSz field size if the I field 626 is one.

Detailed Description Text (167):
For example, in one processor 120 embodiment, an emulation carry flag is defined in addition to the conventional carry flag and associated with the emulation environment. Thus, a conventional add instruction is defined that sets the carry flag and changes the permanent architectural state of the processor 120. An emulation add operation is defined that sets the emulation carry flag and not the conventional carry flag. Various instructions are defined to branch on the basis of the emulation carry flag while other instructions branch on the basis of the conventional carry flag. Thus, the processor 120 is concurrently operational in both a conventional microarchitectural state and in an emulation microarchitectural state. This operation in an emulation environment allows the processor 120 to execute an emulation microsequence and to not change the visible microarchitectural state.

Detailed Description Text (212):
StatMod[3:0] Status group marks indicating which groups of status bits the Op modifies. Bits [3,2,1,0] correspond to {EZF,ECF}, OF, {SF,ZF,AF,PF}, and CF respectively. StatMod is always all zeroes for non-RegOps and is cleared during abort cycles.

Detailed Description Text (355):
There is an additional input from the RUX unit (RUX_NoStatMod) which indicates that the operation being executed there does not modify status flags. A cycle-delayed version, called "NoStatMod," is useful in several situations. The following equations describe this logic.

US Reference Patent Number (25):
5568646

US Reference Patent Number (27):
5619665

CLAIMS:

2. The microprocessor according to claim 1, wherein the RegOp class further includes an operation that modifies one or more status flags, the RegOp class further having defined-usage bit-fields including:

an extension bit-field for specifying the one or more status flags that are modified by the operation; and

a set status bit-field for causing the operation to modify status flags in accordance with the extension bit-field.

5. The microprocessor according to claim 4, wherein the RegOp class further includes an operation that modifies one or more status flags, the RegOp class further having defined-usage bit-fields including:

an extension bit-field for specifying the one or more status flags that are modified by the operation; and

a set status bit-field for causing the operation to modify status flags in accordance with the extension bit-field.

8. The microprocessor according to claim 7, wherein the RegOp class further includes an operation that modifies one or more status flags, the RegOp class further having defined-usage bit-fields including:

an extension bit-field for specifying the one or more status flags that are modified by the operation; and

a set status bit-field for causing the operation to modify status flags in accordance with the extension bit-field.

11. The microprocessor according to claim 10, wherein the RegOp class further includes an operation that modifies one or more status flags, the RegOp class further having defined-usage bit-fields including:

an extension bit-field for specifying the one or more status flags that are modified by the operation; and

a set status bit-field for causing the operation to modify status flags in accordance with the extension bit-field.

14. The microprocessor according to claim 13, wherein the RegOp class further includes an operation that modifies one or more status flags, the RegOp class further having defined-usage bit-fields including:

an extension bit-field for specifying the one or more status flags that are modified by the operation; and

a set status bit-field for causing the operation to modify status flags in accordance with the extension bit-field.

27. A microprocessor according to claim 26, wherein the instruction set RegOp class further includes an operation that modifies one or more status flags, the RegOp class further having defined-usage bit-fields including:

an extension bit-field for specifying the one or more status flags that are modified by the operation; and

a set status bit-field for causing the operation to modify status flags in accordance with the extension bit-field.

29. A microprocessor according to claim 28, wherein the instruction set RegOp class further includes an operation that modifies one or more status flags, the RegOp class further having defined-usage bit-fields including:

an extension bit-field for specifying the one or more status flags that are modified by the operation; and

a set status bit-field for causing the operation to modify status flags in accordance with the extension bit-field.

WEST

Generate Collection | Print

Search Results - Record(s) 1 through 1 of 1 returned.

1. Document ID: NN9403605

L2: Entry 1 of 1 File: TDBD Mar 1, 1994

TDB-ACC-NO: NN9403605

DISCLOSURE TITLE: Real-Time CISC Architecture HW Emulator On A RISC Processor

PUBLICATION-DATA:
IBM Technical Disclosure Bulletin, March 1994, US

VOLUME NUMBER: 37
ISSUE NUMBER: 3
PAGE NUMBER: 605 - 606

PUBLICATION-DATE: March 1, 1994 (19940301)

CROSS REFERENCE: 0018-8689-37-3-605

DISCLOSURE TEXT:

This document contains drawings, formulas, and/or symbols that will not appear on line. Request hardcopy from ITIRC for complete article. - A method for emulating a CISC architecture CPU is disclosed. This method utilizes a RISC architecture microprocessor in association with a CISC to RISC hardware translator. - In the Figure, the hardware translator resides between the RISC subsystem and memory where the CISC macroinstructions reside. The function of the hardware translator is to return RISC instructions that are generated from each CICS macroinstruction. In essence, the RISC microprocessor is acting as a microinstruction engine. - For each RISC microprocessor instruction cache line refill request, the hardware translator calculates the physical address of the corresponding CISC macroinstruction in main memory. After fetching the CISC macroinstruction from memory it performs a table lookup within the hardware translator's memory array based on the CISC opcode. This will point to a series of "canned" RISC microinstructions that emulate the CISC macroinstruction. These RISC microinstructions are then "tailored" for the other components of the CISC macroinstruction. The completed RISC microinstructions are then returned to the RISC subsystem to satisfy the instruction fetch.

Term	Documents
REAL-TIME.TDBD.	615
CISC.TDBD.	14
(REAL-TIME ADJ CISC).TI..TDBD.	1
((REAL-TIME CISC).TI.).TDBD.	1

Display Format: FULL Change Format

Previous Page Next Page

1. Document ID: NN72033074

L1: Entry 1 of 1 File: TDBD Mar 1, 1972

TDB-ACC-NO: NN72033074

DISCLOSURE TITLE: OP Code and Status Handling for Emulation. March 1972.

PUBLICATION-DATA:
IBM Technical Disclosure Bulletin, March 1972, US

VOLUME NUMBER: 14
ISSUE NUMBER: 10
PAGE NUMBER: 3074 - 3076

PUBLICATION-DATE: March 1, 1972 (19720301)

CROSS REFERENCE: 0018-8689-14-10-3074

DISCLOSURE TEXT:

3p. The drawing depicts logic for OP code and status handling when a system, such as the IBM System/370, is provided with a special feature for emulating another data-processing system. The IBM System/370 has an architecture which includes a number of addressable general purpose registers and floating-point registers, and includes a number of instruction formats including an SS format comprised of 48 binary bits, an RR format comprised of 16 binary bits, and an RS format comprised of 32 binary bits. When a base system is provided with an emulation feature, it has become common practice to specify a number of special instructions of the base system capable of performing the logic required by an instruction of the system being emulated. In the System/370, approximately 16 OP codes of the SS format instruction have been set aside for special emulation instructions. - When implementing emulation on a system, special hardware is usually provided for retaining status and control information peculiar to the system being emulated. When one of several jobs in a base data-processing system with multiprogramming capability is emulation, it is necessary that the special status and control information be saved and restored as the emulation job is interrupted and reinstated. For this purpose, a number of the base system registers, such as the addressable floating-point registers, can be utilized for the saving and restoring process. In this manner, the saving and restoring of the special emulator status can be accomplished using the standard interrupt handling capability of the base system control program. - In a system which includes a control store 1, base system microprograms will be contained in a portion 2 and microprograms for executing special emulator instructions will be contained in a portion 3. If it is desired to provide more special emulator instructions than provided by the OP codes set aside in the SS format instructions, the technique described in the drawing can be utilized. In this regard, one of the allowed OP codes of the SS instructions is specified as the Emulator OP code (EMU OP code) contained in bits 0 - 7 of an instruction register 4. A number of flags are provided in bit positions 8 - 15 for the purpose of identifying the system being emulated, when the base system is capable of emulating more than one system. A check is made of the flags to insure that the instruction being decoded is for the system being emulated, as defined by

the emulator microprograms provided in 3. - When the special EMU instruction is recognized by a decoder 5, positions 16 - 47 define a special emulator instruction. Bit positions 16 - 23 provide the special secondary instruction OP code (SEC OP code), which may be either the RR format, providing general register addresses R1 and R2, or the instruction may be of the RS format, providing designation of a general register R1, general register R3, base register B2, and a displacement address value D2. - As indicated previously, emulation may be one of several jobs in the base system. If the emulation job has been interrupted by a signal on line 6, the emulator status information 7 would have been transferred to a number of designated floating-point registers 8 for use by the base system control program for interrupt handling. In this case, a special status valid trigger 9 would be reset to indicate that the emulated system status was not valid. To complete a description of the logic shown in the drawing, a decoder 10 is provided for recognizing the OP code of the secondary instruction contained in the primary EMU instruction. - Decoding of the special EMU instruction will cause one of the following to happen: 1) If the status valid trigger 9 is on, the instruction handling unit will suppress any action of the primary EMU instruction. The indication of status valid will be effective to decode the secondary OP code at 10 and cause access of the microprogram 3 which is necessary to execute the secondary special instruction. - 2) If the status valid trigger 9 is off, the instruction handling unit will cause an OP code microprogram branch to go to the control store section 2 where the primary EMU instruction will be executed. Execution of the primary EMU instruction consists of checking for the proper emulator flags, loading the emulator status from floating-point registers 8 to the emulator status triggers 6, and turning on the status valid trigger 9. The base system microprogram from section 2 will then set the base system instruction counter back to the address of the EMU instruction just executed. The same EMU instruction will then be executed again and the action as described in (1) above will occur.

Generate Collection Print

Term	Documents
OP.TDBD.	632
CODE.TDBD.	7746
STATUS.TDBD.	3877
((OP ADJ CODE) AND STATUS).TI..TDBD.	1
((OP CODE AND STATUS).TI.).TDBD.	1

Display Format: FULL Change Format